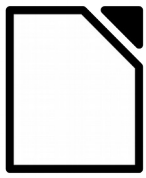


LibreOffice
The Document Foundation

Base

Praktische Beispiele für besondere Problemstellungen in Base

Dokumentationen zu LibreOffice unter de.libreoffice.org

Copyright

Mitwirkende/Autoren

Robert Großkopf

Rückmeldung (Feedback)

Kommentare oder Vorschläge zu diesem Dokument können Sie in deutscher Sprache an die Adresse *discuss@de.libreoffice.org* senden.

Vorsicht

Alles, was an eine Mailingliste geschickt wird, inklusive der E-Mail-Adresse und anderer persönlicher Daten, die die E-Mail enthält, wird öffentlich archiviert und kann nicht gelöscht werden. Also, schreiben Sie mit Bedacht!

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 1.2.2015. Basierend auf der LibreOffice Version 4.4.

Anmerkung für Macintosh Nutzer

Einige Tastenbelegungen (Tastenkürzel) und Menüeinträge unterscheiden sich zwischen der Macintosh Version und denen für Windows- und Linux-Rechnern. Die unten stehende Tabelle gibt Ihnen einige grundlegende Hinweise dazu. Eine ausführlichere Aufstellung dazu finden Sie in der Hilfedatei des jeweiligen Moduls.

Windows/Linux	***entspricht am Mac***	***Effekt***
Menü-Auswahl **Extras → Optionen**	**LibreOffice → Einstellungen**	Zugriff auf die Programmoptionen
Rechts-Klick	*Control*+Klick	Öffnen eines Kontextmenüs
Ctrl (Control) oder *Strg* (Steuerung)	⌘ *(Command)*	Tastenkürzel in Verbindung mit anderen Tasten
F5	*Shift*+⌘+*F5*	Öffnen des Dokumentnavigator-Dialogs
F11	⌘+*T*	Öffnen des Formatvorlagen-Dialogs

Inhalt

Wozu Datenbankbeispiele?

Für viele Nutzer ist die Hürde beim Erstellen von Datenbanken recht hoch. Schon die Erstellung von Tabellen, ihre Verknüpfung sowie die Beschickung in einfachen Formularen stellt häufig ein Problem dar.

Diese Probleme sollte allerdings das LO-Handbuch «Erste Schritte» sowie das Base-Handbuch bewältigen helfen. Diese Beispiele ergänzen vor allem das Base-Handbuch. Die Beispiele sind meist speziell auf eine Problemstellung hin ausgerichtet. Meist stellen sie, im Gegensatz zu den Mediendatenbanken für das Base-Handbuch, keine komplette Lösung für eine Datenbank dar. Die Reihenfolge der Beispiele ist nicht vom Einfachen zum Komplizierten gewählt. Jedes Kapitel steht für sich. Es kann aber sein, dass in nachfolgenden Kapiteln auf bestimmte Konstruktionen des vorangegangenen Kapitel Bezug genommen wird. In diesem Falle also bitte einfach die entsprechende Stelle aufsuchen.

Diese Beschreibung deckt bisher nicht alle Beispieldatenbanken ab, die mit dem Base-Handbuch weiter gegeben werden. Die Beschreibung soll daher ständig weiter vervollständigt werden.

Zeitmessung in einer Datenbank

Zeitmessungen könnten z. B. für die Durchführung von Sportveranstaltungen genutzt werden. Gedacht ist hier nicht an große Veranstaltungen, wo so etwas sowieso elektronisch erfolgt. Vielmehr könnte es um Veranstaltungen im Schulbereich gehen.

Der PC ersetzt hier die Stoppuhr und das Niederschreiben der Ergebnisse. Er kann außerdem auch die Auswertung erstellen sowie Urkunden drucken. Im Vordergrund soll hier die Funktion der Stoppuhr stehen.

Um mit einer Datenbank auch Zeitmessungen durchführen zu können werden in geringem Umfang Makrokenntnisse gebraucht.

Tabellen

Die Datenbank besteht im sparsamsten Aufbau aus zwei Tabellen.

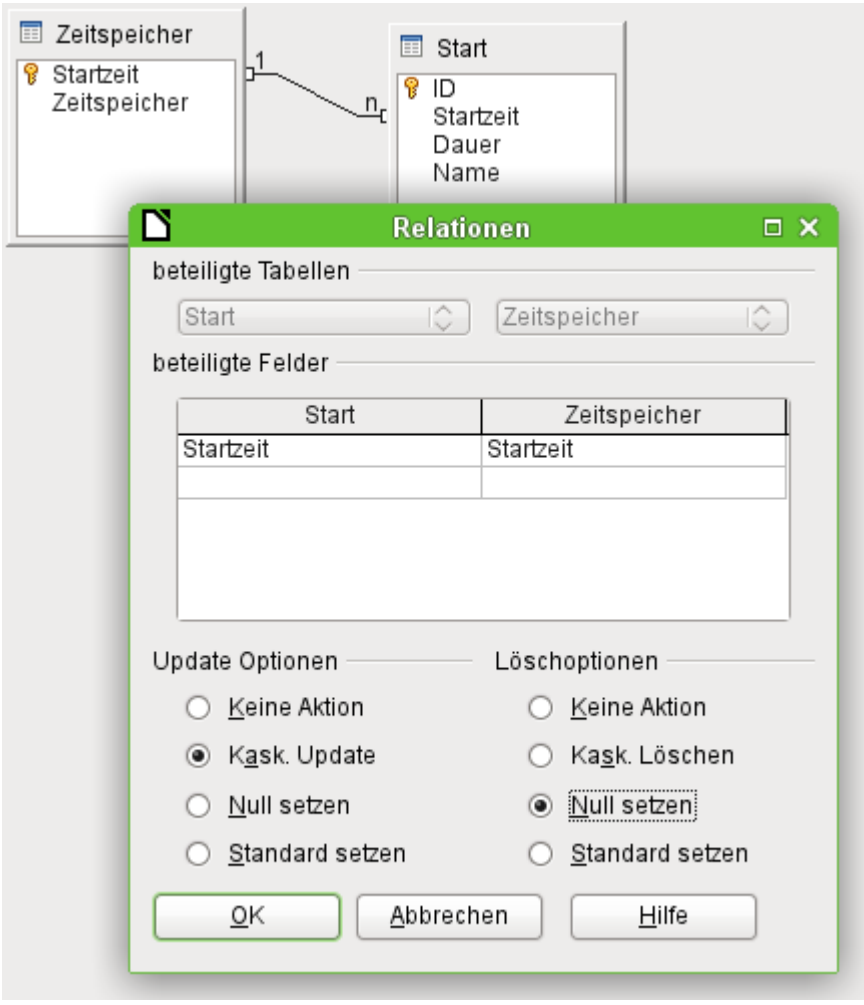
In der Tabelle "Start" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Startzeit	Datum/Zeit	Hier wird Tag und Uhrzeit des vorgesehenen Startes festgehalten. Personen, die zusammen starten, haben die gleiche Startzeit.
Name	Text	Der Name der Person, die startet. Bei Schulwettkämpfen wären hier Zusätze wie Klasse, Schule oder gegebenenfalls auch Geburtsdaten notwendig.
Dauer	Integer	Die Dauer wird in Sekunden oder Millisekunden gemessen. Es handelt sich dabei um einen Zahlenwert ohne Nachkommastellen, keinen Wert wie bei einer Uhrzeit.
Zeit (nachträglich eingefügt)	Datum/Zeit	Die Zeitdauer wird hier als Zeit im Format Minuten:Sekunden,Hundertstelsekunden aus dem Formular «Einzelmessung_Zeitfeld» übernommen. Der Datumsteil wird durch ein Makro hinzugefügt.

In der Tabelle "Zeitspeicher" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
Startzeit	Datum/Zeit	Die Startzeit ist Primärschlüssel dieser Tabelle. Sie ist gleichzeitig Fremdschlüssel der Tabelle "Start". Damit wird für alle Personen, die die gleiche Startzeit haben, auch bei der Durchführung der einheitliche Start festgelegt.
Zeitspeicher	Integer	Hier wird ein interner Basic-Wert für die tatsächliche Startzeit abgespeichert. Der Wert wird erst dann eingefügt, wenn der Button «Start» betätigt wurde.

Beide Tabellen werden miteinander in den Beziehungen verbunden. Es muss also eine Startzeit festgelegt werden, bevor eine Person in die Startliste übertragen werden kann.



Die Beziehung zwischen den Tabellen wird mit einem rechten Mausklick auf die Verbindungslinie genauer definiert. Wird im "Zeitspeicher" eine "Startzeit" geändert, so werden alle "Startzeit"-Einträge in der Tabelle "Start" entsprechend angepasst. Dies ist über die **Update Optionen → Kask. Update** gewährleistet. Wird eine "Startzeit" aus der Tabelle "Zeitspeicher" gelöscht, so sollen Einträge, die sich aus der Tabelle "Start" genau auf die gelöschte "Startzeit" beziehen, geleert werden. Dies wird über die **Löschoptionen → Null setzen** erreicht.

Die tatsächlichen Startzeiten müssen nicht mit der eingetragenen Startzeit übereinstimmen. Auch wenn eine eingetragene Startzeit z. B. 10:00 Uhr ist, kann tatsächlich um 15:13 Uhr gestartet werden und trotzdem ein genaues Messergebnis erwartet werden.

Zusätzlich gibt es eine Tabelle "Filter". Diese Tabelle dienst nur dazu, einen Filterwert aus einem Hauptformular an ein Unterformular weiter zu geben.

Feldname	***Feldtyp***	***Beschreibung***
ID	Ja/Nein	Die Filtertabelle soll nur einen Datensatz abspeichern. So lässt sie sich auch problemlos in Abfragen als Datenquelle für eine Unterabfrage nutzen.
Start_ID	Integer	Hier wird der Primärschlüssel der Tabelle "Start" gespeichert. Das ist im Formular der Wert für den Zieleinlauf. Die Eingabe einer Startnummer sollte beim Abspeichern die Zeit mit speichern können. Dies soll über die Filtertabelle erreicht werden.

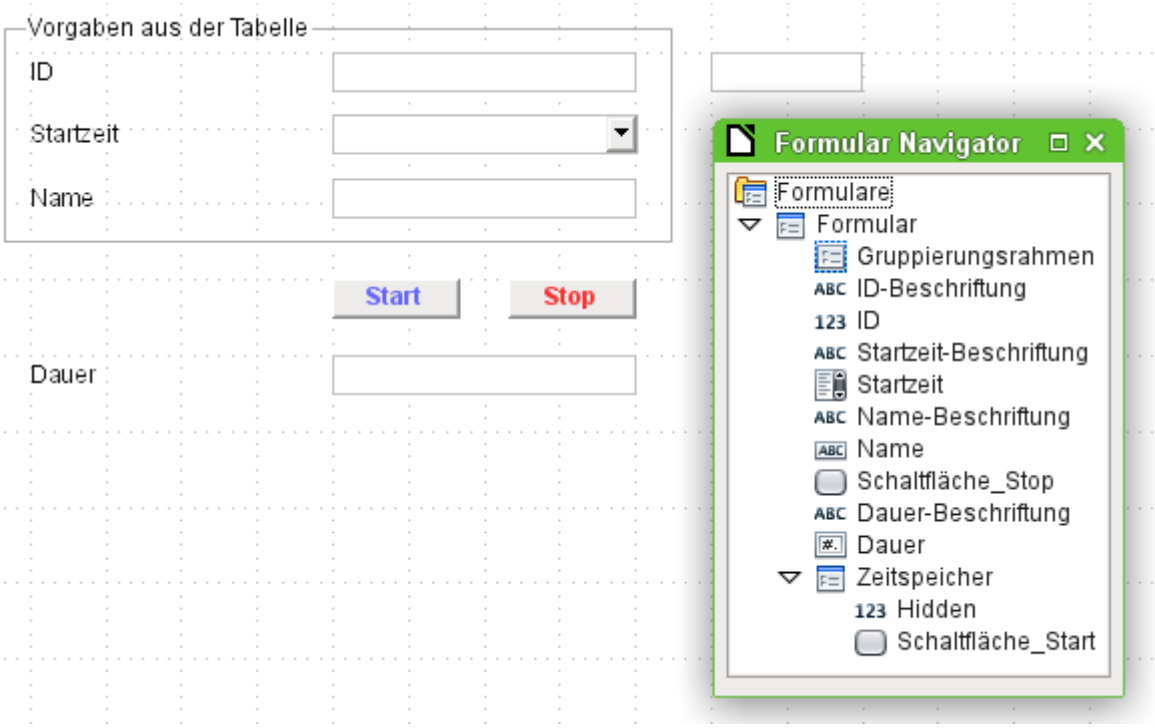
Formulare

Die Messung erfolgt innerhalb von Formularen. Ein Formular zeigt nur die Messung eines Wertes. In den folgenden Formularen werden dann auch Mitbewerber angezeigt. Schließlich wird eine komplette Zeitmessung für ein Rennen ausgestellt, bei dem gleichzeitig gestartete Teilnehmer in eine Rangliste eingeordnet werden. Hier zuerst einmal die einfachste Variante eines Formulars mit einfachen Feldern.

Das Formular «Einzelmessung»

Das Formular sieht erst einmal recht karg aus. Die ursprünglich geplante Startzeit ist zusammen mit dem Datum als Zeitstempel abgespeichert. Da für die Datenbank über die oben genannten 2 Tabellen auch andere Funktionen gewährleistet werden sollen, kann hier nur dann eine Startzeit/Startdatum angegeben werden, wenn die entsprechende Zeit bereits in der anderen Tabelle vorhanden ist. Deshalb ist für die Startzeit ein Listenfeld vorgesehen. Das Listenfeld liest die "Startzeit" aus der Tabelle "Zeitspeicher" und trägt genau diesen Wert in die Tabelle "Start" als "Startzeit" ein. Für die Messung ist diese Zeit wegen der Abspeicherung der Startzeit in der zweiten Tabelle zwingend notwendig.

Start startet die Zeitmessung. Mit Stop wird die Zeitmessung beendet und die gemessene Zeit in dem Feld «Dauer» abgespeichert.



Die Entwurfsansicht des Formulars gibt etwas genauer Aufschluss darüber, wie die Zeit ermittelt wird.

In dem Formular befindet sich ein Unterformular mit dem Namen «Zeitspeicher». Dieses Unterformular ist verbunden mit der Tabelle "Zeitspeicher". Das als unsichtbar definierte Element «Hidden» ist mit dem Datenfeld "Zeitspeicher" verbunden. Formular und Unterformular sind über die "Startzeit" verbunden.

Der Button Start liegt in dem Unterformular. Durch den Formularwechsel vom Hauptformular zum Unterformular wird beim Betätigen des Start-Buttons also der Inhalt des Hauptformulars abgespeichert. Wird Start betätigt, so schreibt ein Makro die interne Startzeit in das versteckte Feld.

Der Button Stop könnte auch im Unterformular liegen. Die Abspeicherung des gemessenen Wertes wird schließlich per Makro erledigt. Wird Stop betätigt, so wird über das Makro die Differenz zwischen dem Wert in dem versteckten Feld und dem momentanen Wert ermittelt und in das Feld "Dauer" übertragen. Die Messung erfolgt in Millisekunden.

Makros für das Formular «Einzelmessung»

Der Button Start startet über das Ereignis «Aktion ausführen» das Makro «Zeitmessung_Start».

```
SUB Zeitmessung_Start
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM lZeit AS LONG
```

Zuerst werden die Variablen benannt und mit den ihnen entsprechenden Typen versehen. Bis auf eine Variable sind alle vom Typ «Object». Die Zeitangabe wird hingegen als Zahl gespeichert. Der Typ «Long» entspricht dem größten Zahlentyp ohne Nachkommastellen in StarBasic. Er entspricht vom Umfang her einem Integer-Feld der Datenbank.

```
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oSubForm = oForm.getByName("Zeitspeicher")
    oFeld = oSubForm.getByName("Hidden")
```

Das Feld «Hidden» wird aufgesucht. Es soll den internen Wert für die Startzeit speichern. Das Feld wird über das Formular und das darin liegende Unterformular erreicht. Die angegebenen Namen entsprechen den Namen, die im Formularnavigator sichtbar sind.

```
        'lZeit = Timer() ' Gibt die Systemzeit in Sekunden an
        lZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
```

Die Zeitspeicherung kann entweder in Sekunden oder in Millisekunden vorgenommen werden. Hier ist die Variante für die Millisekunden ausgewählt. Die Systemzeit wird ausgelesen.

```
        oFeld.BoundField.UpdateInt(lZeit)
        oSubForm.UpdateRow()
    END SUB
```

Anschließend wird die Systemzeit in das Feld «Hidden» übertragen. **`BoundField`** bedeutet hier das an dieses Feld gebundene Feld der Tabelle, in der der Wert abgespeichert werden soll. **`UpdateInt`** heißt, dass dieses Feld einen Integer-Wert abspeichern wird. Es ist in der Tabelle jetzt das Feld auf einen neuen Integer-Wert gesetzt worden. Dieser Wert ist jetzt in dem Feld der Tabelle enthalten und muss mit **`UpdateRow`** über das entsprechende Formular noch abgespeichert werden.

Der Button Stop startet über das Ereignis «Aktion ausführen» das Makro «Zeitmessung_Ende».

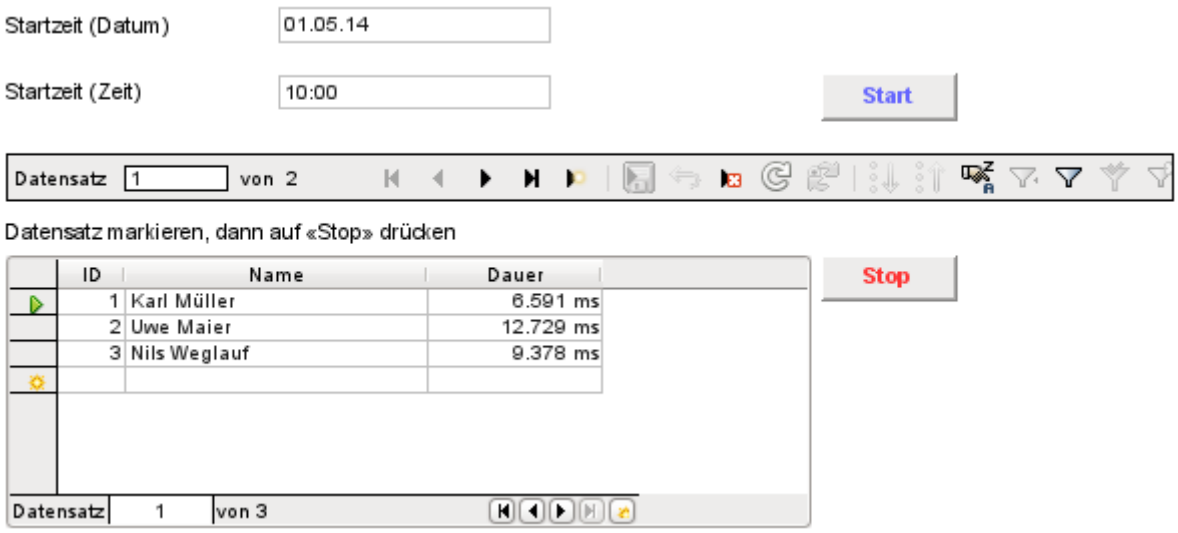
```
    SUB Zeitmessung_Ende
        DIM oDoc AS OBJECT
        DIM oDrawpage AS OBJECT
        DIM oForm AS OBJECT
        DIM oFeld AS OBJECT
        DIM oFeld1 AS OBJECT
        DIM lZeit AS LONG
        DIM lZeit1 AS LONG
        oDoc = thisComponent
        oDrawpage = oDoc.drawpage
        oForm = oDrawpage.forms.getByName("Formular")
        oSubForm = oForm.getByName("Zeitspeicher")
        oFeld = oSubForm.getByName("Hidden")
        oFeld1 = oForm.getByName("Dauer")
```

Die Ansprache der Felder im Formular ist gleich. Jetzt wird allerdings noch ein zusätzliches das Feld «Dauer» angesteuert, in dem schließlich die gemessene Zeit gespeichert werden soll.

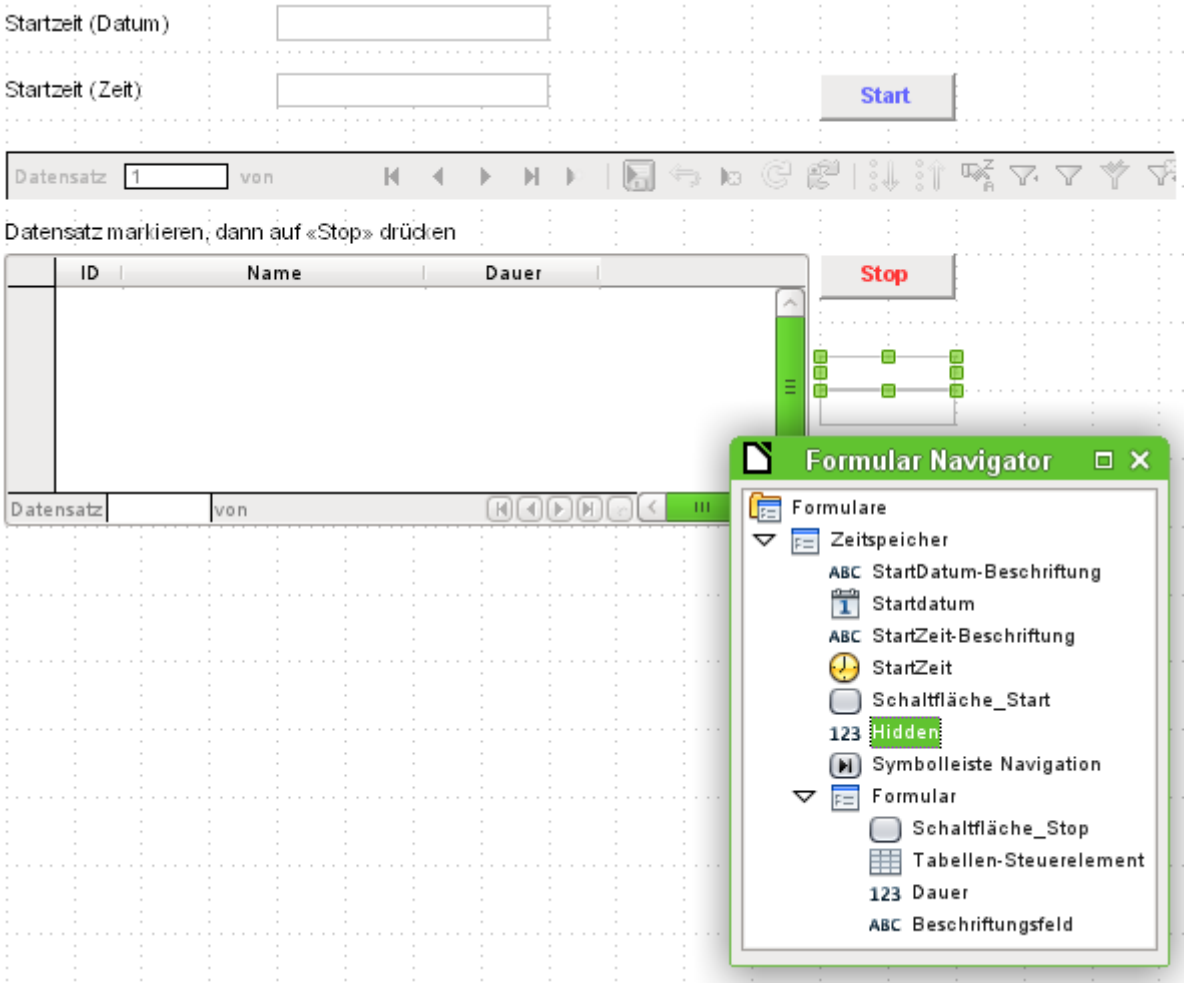
```
        'lZeit1 = Timer()   ' Gibt die Systemzeit in Sekunden an
        lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
        lZeit = oFeld.getCurrentValue
        oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
        oForm.UpdateRow()
    END SUB
```

Es wird die aktuellen Zeit, wie im vorhergehenden Makro, ausgelesen. Dann wird der gespeicherte Startwert aus dem Feld «Hidden» ausgelesen. Der Startwert wird von dem aktuellen Wert subtrahiert. Das Ergebnis der Subtraktion wird in das Feld «Dauer» übertragen und anschließend über das entsprechende Formular abgespeichert.

Das Formular «Startzeit_Gruppe»



Dieses Formular ermöglicht es, neue Startzeiten einzugeben. Den Startzeiten werden im Unterformular die Personen zugeordnet, die zu der jeweiligen Zeit an den Start gehen wollen. Hier ist z.B. eine Gruppe von 3 Personen am 1.5.14 um 10:00 Uhr gemeinsam gestartet. Die Zeit, die die einzelnen Personen für die Strecke benötigt haben, wird unter «Dauer» erfasst.



Im Hauptformular wird auf die Tabelle "Zeitspeicher" zugegriffen. In ihr wird die "Startzeit" und auch die systeminterne Startzeit (in dem Feld "Zeitspeicher") abgespeichert. Die systeminterne Startzeit

braucht nicht für den Nutzer sichtbar zu erscheinen. Das Feld mit der Bezeichnung «Hidden» ist deshalb einfach irgendwo auf dem Formular untergebracht.

Über Start wird dieses Feld mit der systeminternen Startzeit versorgt.

Die Tabelle "Start" liegt jetzt im Unterformular. In ihr wird auch der Zieleinlauf des Rennens gespeichert. Verbindungsglied zwischen Hauptformular und Unterformular ist das Feld "Startzeit". Die Zeit, die die jeweilige Person für die Strecke benötigt hat, wird in dem außerhalb des Tabellenkontrollfeldes versteckten Feld «Dauer» abgespeichert. Dieses Feld ist über Makros erst einmal einsichtiger zu erreichen als ein Feld innerhalb des Tabellenkontrollfeldes. Gleichzeitig zeigt es auch, dass Werte des aktuellen Datensatzes des Tabellenkontrollfeldes außerhalb des Tabellenkontrollfeldes angezeigt oder eingegeben werden können. So etwas kann z.B. auch für Textfelder mit viel Inhalt oder auch für abgespeicherte Bilder sinnvoll sein.

Makros für das Formular «Startzeit_Gruppe»

```
SUB Zeitmessung_Massenstart
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM lZeit AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Zeitspeicher")
    oFeld = oForm.getByName("Hidden")
    'lZeit = Timer() ' Gibt die Systemzeit in Sekunden an
    lZeit = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    oFeld.BoundField.UpdateInt(lZeit)
    IF oForm.RowInserted() THEN
        oForm.InsertRow()
    ELSE
        oForm.UpdateRow()
    END IF
END SUB
```

Da der Formularaufbau (Hauptformular zu Unterformular) umgedreht wurde müssen die Felder entsprechend anders angesprochen werden. Im Hauptformular wird jetzt der Start eingetragen. Es wäre zwar sinnlos, einen Start ohne entsprechend zugeordnete Personen ablaufen zu lassen. Trotzdem soll hier einfach über Start auch abgesichert werden, dass es bei einer Neueingabe von Daten nicht zu einer Fehlermeldung kommt. Aus diesem Grunde wird abgefragt, ob es sich um eine eingefügten Datensatz handelt (**`oForm.RowInserted`**). Ist der Datensatz neu einzufügen, dann müssen die Daten an die Tabelle mit **`oForm.InsertRow`** weitergegeben werden. Ansonsten handelt es sich um eine Datenänderung, also **`oForm.UpdateRow`**.

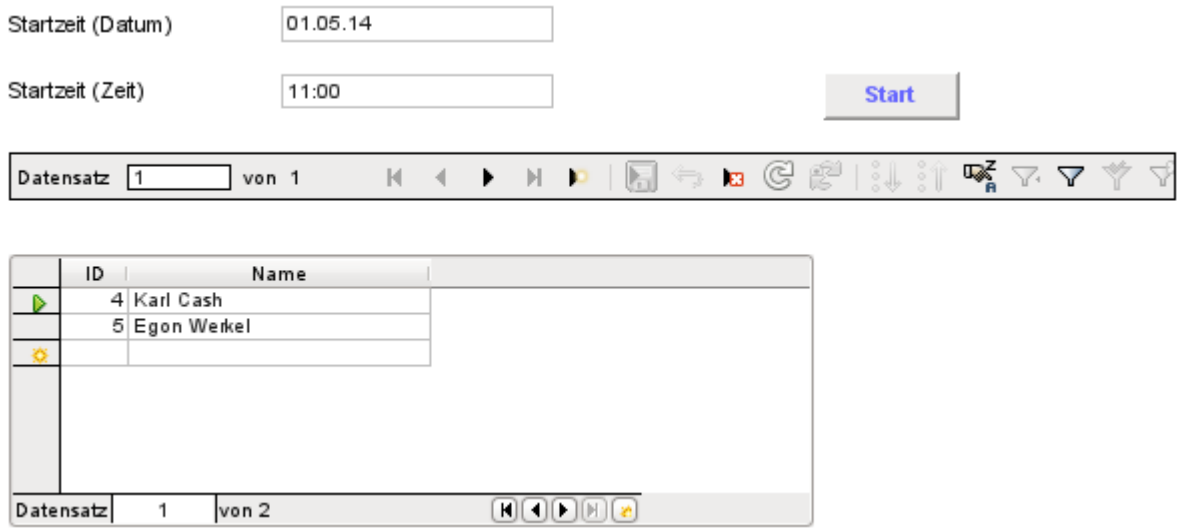
```
SUB Zeitmessung_Ende_Massenstart
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM lZeit AS LONG
    DIM lZeit1 AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Zeitspeicher")
    oSubForm = oForm.getByName("Formular")
    oFeld = oForm.getByName("Hidden")
    oFeld1 = oSubForm.getByName("Dauer")
    'lZeit1 = Timer()    ' Gibt die Systemzeit in Sekunden an
    lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    lZeit = oFeld.getCurrentValue
    oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
    oSubForm.UpdateRow()
END SUB
```

Die Zeitmessung wird genau so wie im ersten Formular gestoppt. Unterschied zwischen den Makros ist hier lediglich die entsprechende Abfrage der Felder. Das Feld «Dauer» liegt dabei im Unterformular. Um das Feld innerhalb des Tabellenkontrollfeldes anzusprechen wäre noch eine zusätzliche Zeile notwendig gewesen. Der Zugriff über Namen kann dabei nur funktionieren, wenn unterschiedliche Namen vergeben wurden. Ansonsten müssen Felder über **`getByIndex`** angesprochen werden.

Das Formular «Nur_Start»

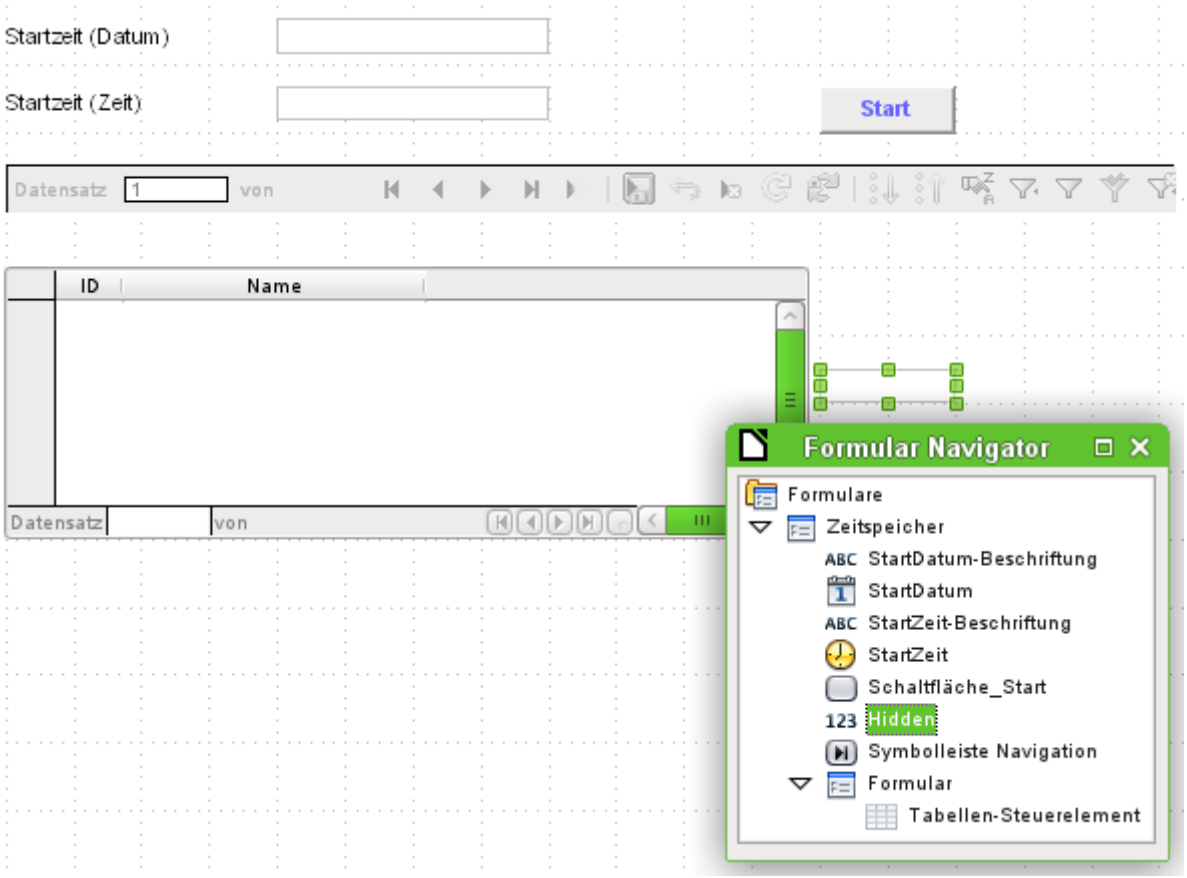
Bei den bisherigen Konstruktionen kann es ohne weiteres passieren, dass ein Start mehrmals hintereinander eingegeben, also die Zeit nicht korrekt gemessen wird. Dies lässt sich vermeiden, wenn nach dem erfolgten Start ein erneutes Beschreiben des Start-Wertes unmöglich gemacht wird.

Aus dem folgenden Formular verschwinden nach dem Start die Datensätze der Personen, die zur gleichen Startzeit unterwegs sind.

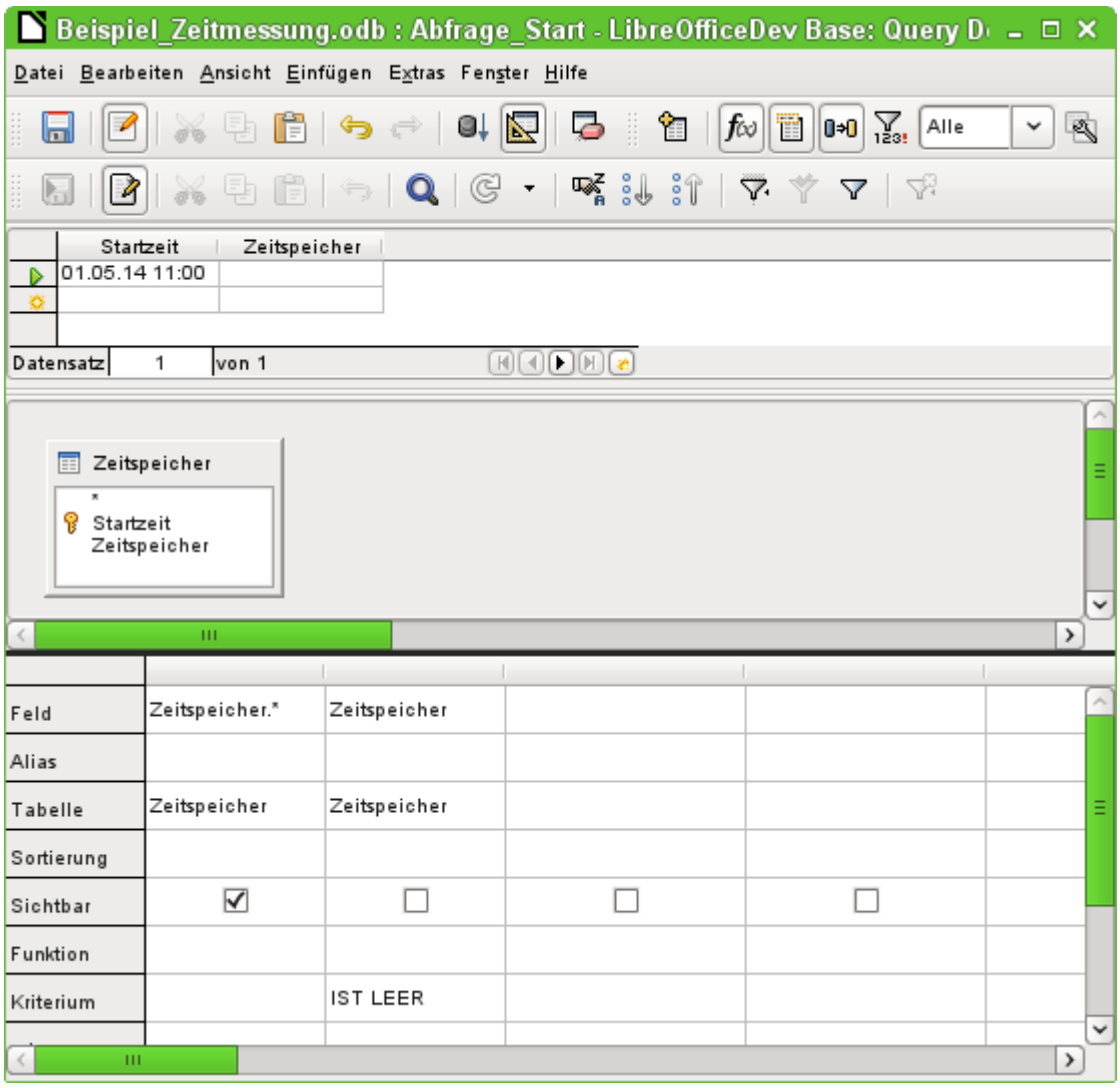


Die Formularansicht ist gegenüber dem Formular «Startzeit_Gruppe» etwas reduziert. Natürlich gibt es in einem reinen Start-Formular keinen Stop-Button. Außerdem werden nur die Namen der Personen aufgelistet, die zu der entsprechenden Zeit am Start stehen müssen. Die «Dauer», die sie unterwegs sind, wird in diesem Formular nicht erfasst.

Auch der Formularnavigator zeigt nur eine reduzierte Ansicht des Formulars «Startzeit_Gruppe». Elemente für den Zieleinlauf fehlen.



Als Grundlage für das Formular «Zeitspeicher» fungiert hier allerdings nicht die Tabelle "Zeitspeicher", sondern eine Abfrage:



Die Abfrage bezieht sich auf die Tabelle "Zeitspeicher". Sie zeigt alle Felder dieser Tabelle an (im ersten Feld des grafischen Editors steht: Zeitspeicher.*). Da sich die Abfrage nur auf diese eine Tabelle bezieht ist sie so auf jeden Fall editierbar.

Im zweiten Feld wird definiert, dass "Zeitspeicher" auf jeden Fall leer sein soll, wenn der Datensatz angezeigt wird. Alle Felder mit ausgefülltem "Zeitspeicher" erscheinen also nicht. Die so formulierte Abfrage lautet schließlich:

```
SELECT * FROM "Zeitspeicher" WHERE "Zeitspeicher" IS NULL
```

Diese recht einfache Abfrage hätte auch direkt im Formular «Zeitspeicher» als Filterwert eingegeben werden können:

rechte Maustaste über dem Formularnamen «Zeitspeicher» → Eigenschaften → Daten → Filter

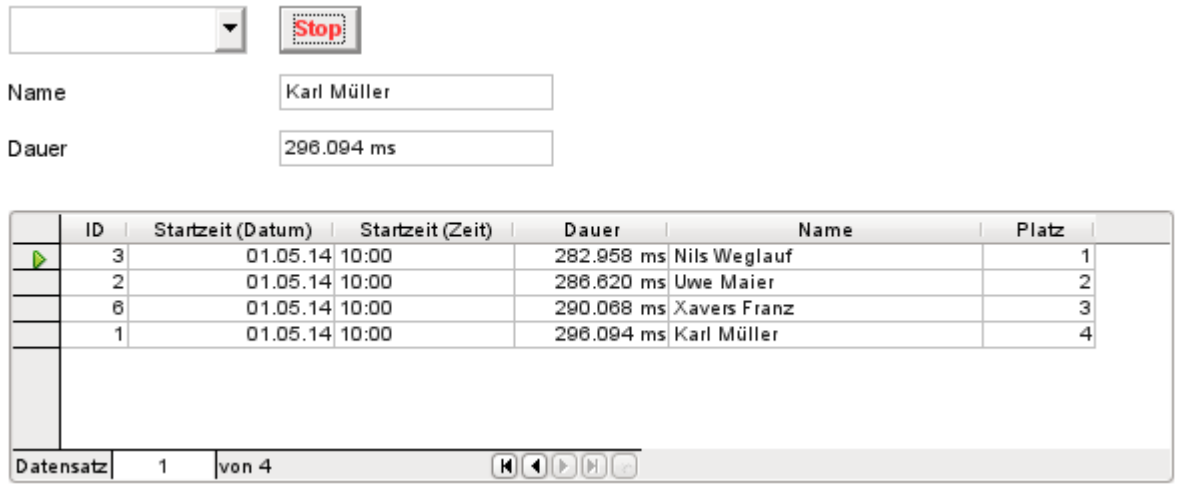
In das freie Feld kann eingetragen werden: **`("Zeitspeicher" IS NULL)`**

Makros für das Formular «Nur_Start»

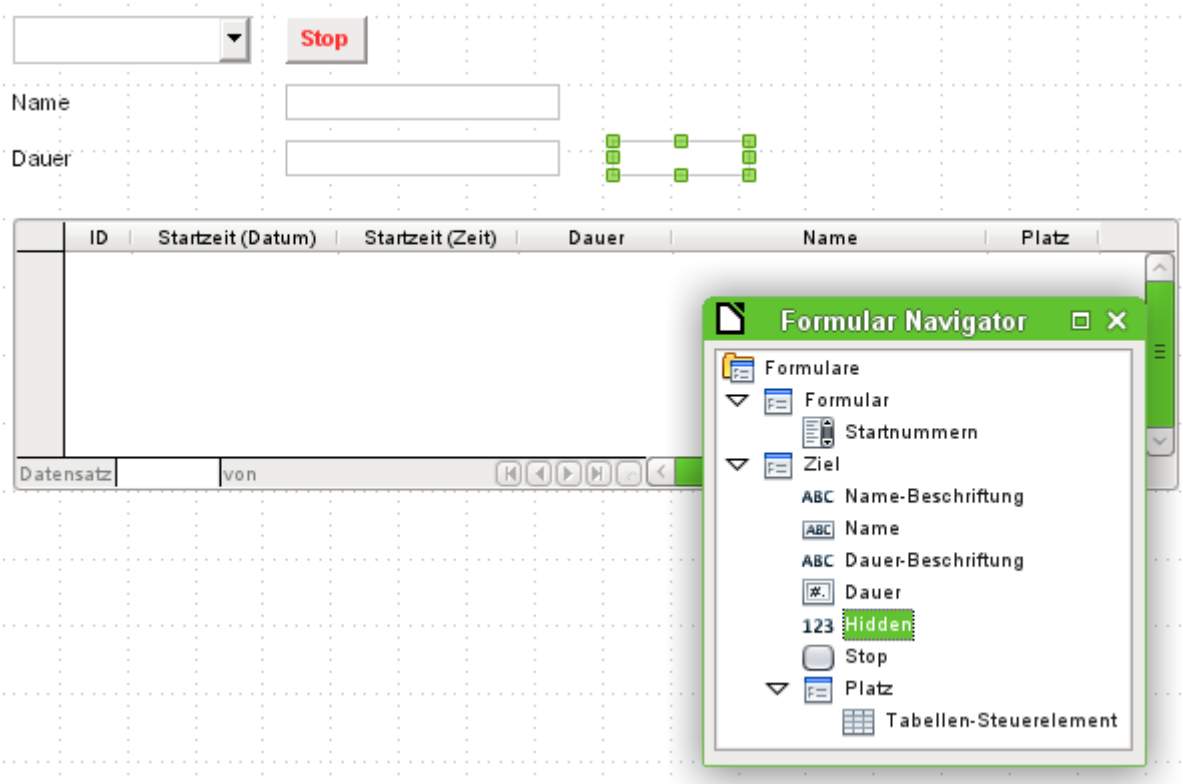
Das Formular «Nur_Start» nutzt das Makro «Zeitmessung_Massenstart», das bereits im vorangegangenen Formular über den Button Start ausgelöst wurde.

Das Formular «Nur_Ziel»

Dieses Formular muss wesentlich mehr Funktionen erfüllen als die vorhergehenden Formulare. Es soll dazu dienen, nur nach Auswahl der Startnummer den Zieldurchgang zu regeln. Dabei soll es anzeigen, welche Zeit die gerade das Ziel passierte Person benötigt hat. Außerdem soll noch die Einordnung in Form einer Rangliste erfolgen.



Hier ist gerade Karl Müller als 4. Person ins Ziel gekommen. Die Startnummer ("ID") von Karl Müller wurde aus dem Listenfeld direkt neben dem Button Stop ausgewählt, als sich Karl Müller dem Ziel näherte. Stop wurde gedrückt, der Name zu der Startnummer und die Zeit wurde angezeigt, gleichzeitig die Positionierung in dem Feld der Personen, die zur gleichen Zeit gestartet waren. Das Listenfeld zeigt die Startnummer «1» von Karl Müller jetzt nicht mehr an, da Karl Müller durchs Ziel gekommen ist.



Der Formularnavigator zeigt gleich drei Formulare. Das erste Formular «Formular» enthält nur ein Listenfeld für die «Startnummern». Es ist von den anderen Formularen getrennt, liegt also neben dieses Formularen auf der Benutzeroberfläche. Die Anzeige von «Formular» soll keine Auswirkung

auf die anderen Formulare haben. Schließlich ist ein Wert, der im Listenfeld «Startnummern» noch verzeichnet ist, nicht in den anderen Formularen zu finden.

Das Formular «Formular» hat als Datengrundlage die Tabelle "Filter" angegeben. Das Formular ist nur zur Änderung von Daten vorgesehen. Es sollen keine Daten gelöscht werden und keine neuen Datensätze angelegt werden. Die Tabelle "Filter" besteht so nur aus einem Datensatz.

Das Listenfeld «Startnummer» zeigt den gleichen Wert an, den es auch an die darunterliegende Tabelle weitergibt. Die Datenquelle für das Listenfeld ist auf SQL eingestellt.

```
SELECT "Start"."ID", "Start"."ID"
   FROM "Start", "Zeitspeicher"
   WHERE "Start"."Startzeit" = "Zeitspeicher"."Startzeit"
      AND "Zeitspeicher"."Zeitspeicher" IS NOT NULL
      AND "Start"."Dauer" IS NULL
```

Diese Abfrage stellt zweimal das gleiche Feld "Start"."ID". Das ist notwendig, wenn in den **Eigenschaften des Listenfeldes → Daten** Gebundenes Feld = 1 steht. In neueren Versionen von LO kann mit der Einstellung Gebundenes Feld = 0 auch nur ein Feld angegeben werden.

Die Abfrage stellt zuerst einmal die Beziehung zwischen den Tabellen "Start" und "Zeitspeicher" her. Dann muss die Bedingung erfüllt sein, dass "Zeitspeicher"."Zeitspeicher" nicht leer ist. Das bedeutet, dass eben ein Start bereits erfolgt ist. Und schließlich sollen nur die Datensätze erscheinen, bei denen in "Start"."Dauer" bisher kein Eintrag existiert. Datensätze von Personen, deren Zeit schon gemessen wurde, fallen also auch heraus.

Das Formular «Ziel» hat als Datengrundlage eine Abfrage, und zwar "Abfrage_Ziel". Diese Abfrage wird zusätzlich über **Formular-Eigenschaften → Daten → Filter** eingeschränkt, so dass nur der aktuelle Datensatz angezeigt wird, der dem Eintrag in der Tabelle "Filter" entspricht:

```
("a"."ID" = (SELECT "Start_ID" FROM "Filter" WHERE "ID" = TRUE))
```

Das Unterformular «Platz» greift auf die gleiche Abfrage zu. Hier ist die Filterung dann allerdings anders angelegt. Unter **Formular-Eigenschaften → Daten → Filter** steht:

```
( "a"."Dauer" IS NOT NULL )
```

Es werden nur die Werte aus der Abfrage angezeigt, die einen Eintrag im Feld "a"."Dauer" haben. Es werden also nur die Personen angezeigt, die auch tatsächlich mit einer entsprechend vermerkten Zeit das Ziel erreicht haben.

Unter **Formular-Eigenschaften → Daten → Sortierung** steht:

"Platz" ASC

Dies bedeutet, dass Datensätze nach der erreichten Platzierung, vermerkt im Feld "Platz", angezeigt werden sollen.

Die Abfrage, auf die sich die Formulare «Ziel» und «Platz» beziehen, sieht im SQL-Code so aus:

```
SELECT "a".*,
   ( SELECT "Zeitspeicher" FROM "Zeitspeicher" WHERE "Startzeit" =
   "a"."Startzeit" ) AS "Zeitspeicher",
   ( SELECT COUNT( "ID" ) FROM "Start" WHERE "Dauer" <= "a"."Dauer"
   AND "Startzeit" = "a"."Startzeit" ) AS "Platz"
   FROM "Start" AS "a"
   WHERE "Zeitspeicher" IS NOT NULL
```

Bei dieser Abfrage handelt es sich um eine korrelierte Unterabfrage. Innerhalb der Abfrage wird auf Werte der aktuellen Abfrage Bezug genommen. Um so eine korrelierte Abfrage zu erstellen muss der Tabelle, auf der die äußere Abfrage beruht, ein Alias zugewiesen werden: **`"Start" AS "a"`**.

Mit **`"a".*`** werden alle Felder der Tabelle "Start" angezeigt. Die Abfrage erfüllt also eine wichtige Voraussetzung dafür, dass sie editierbar ist, indem der Primärschlüssel aus der Tabelle "Start" auch in der Abfrage auftaucht.

Der **"Zeitspeicher"** wird hier durch eine Unterabfrage abgerufen, da sonst die Abfrage nicht mehr editierbar wäre. Die Unterabfrage sucht in der Tabelle **"Zeitspeicher"** nach dem Wert für das Feld **"Zeitspeicher"**. Der Wert muss mit der "Startzeit" des aktuellen Datensatzes von **"Start"**, also **"a"."Startzeit"**, übereinstimmen. Das trifft jeweils auf höchstens einen Datensatz zu und kann also in einer Unterabfrage verwandt werden. Unterabfragen, die mehrere Datensätze ergeben, können in Feldern einer Abfrage nicht genutzt werden.

Die zweite korrelierte Unterabfrage ermittelt die aktuelle Platzierung der im jeweiligen Datensatz repräsentierten Person. Es werden alle Einträge im Feld **"ID"** gezählt. Die **"Dauer"** dieser Einträge muss allerdings kleiner oder gleich der Dauer des aktuellen Datensatzes, also **"a"."Dauer"** sein. Außerdem muss die **"Startzeit"** gleich der Startzeit des aktuellen Datensatzes also **"a"."Startzeit"** sein.

Mit dieser Abfrage "Abfrage_Ziel" lässt sich der gesamte Zieldurchlauf mit Platzierungen darstellen. Wird für die Dauer des Rennens kein Wert im Millisekunden-Bereich benötigt, sondern z.B. nur im Zehntelsekunden-Bereich, so kann in so einer Abfrage entsprechend nachgebessert werden.

Makro für das Formular «Nur_Ziel»

```
SUB Zeitmessung_nur_Ziel
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oSubForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM lZeit AS LONG
    DIM lZeit1 AS LONG
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oForm2 = oDrawpage.forms.getByName("Ziel")
    oSubForm2 = oForm2.getByName("Platz")
    oFeld = oForm2.getByName("Hidden")
    oFeld1 = oForm2.getByName("Dauer")
    oForm.UpdateRow()
    oForm2.Reload()
    oForm2.Last()
    'lZeit1 = Timer()   ' Gibt die Systemzeit in Sekunden an
    lZeit1 = getSystemTicks() ' Gibt die Systemzeit in Millisekunden an
    lZeit = oFeld.getCurrentValue
    oFeld1.BoundField.UpdateInt(lZeit1-lZeit)
    oForm2.UpdateRow()
    oForm.getByName("Startnummern").Refresh()
END SUB
```

Zuerst werden die verschiedenen Formularfelder, wie vorher bereits, zugänglich gemacht. Dann wird bei Betätigung des Buttons Stop das Formular "Formular" mit **oForm.UpdateRow** abgespeichert. Anschließend wird das Formular "Ziel" mit **oForm2.Reload** neu geladen. Da hier der Datensatzzeiger in LibreOffice 4.3.0.0 Beta1 (aus nicht nachvollziehbaren Gründen) nicht auf den letzten Datensatz springt, sondern einen neuen Datensatz erstellen will, wird zusätzlich der Datensatzzeiger des Formulars auf den letzten (einzigen) Datensatz eingestellt: **oForm2.Last**.

Das Auslesen und Eintragen des Zeitmesswertes erfolgt wie bereits bei den anderen makros erklärt. Gleiches gilt auch für die entsprechende Abspeicherung.

Schließlich wird noch das Listenfeld für die Startnummern neu eingelesen, da ja jetzt genau der Datensatz fehlt, den es vorher angezeigt hatte:
oForm.getByName("Startnummern").Refresh().

Das Formular «Einzelmessung_Zeitfeld»

Dieses Formular unterscheidet sich von dem Formular «Einzelmessung» äußerlich nur gering. Statt der Angabe einer Zeit in Millisekunden erfolgt hier die Zeitangabe in der Schreibweise **`Minuten:Sekunden,Hundertstelsekunden`**.

Um diese Zeitangabe entsprechend speichern zu können muss zuerst einmal der Tabelle «Start» ein zusätzliches Feld «Zeit» hinzugefügt werden. Dieses Feld muss als «Datum/Zeit [TIMESTAMP]» - Feld definiert werden. Die reinen Zeitfelder sind nicht in der Lage, Zeiten im Bereich unterhalb einer Sekunde zu speichern. Leider sind aber auch die TIMESTAMP-Felder von der Standardeinstellung in Base nicht dafür vorgesehen. Hier muss entsprechend über SQL nachgeholfen werden. In **Extras → SQL** ist einzugeben:

```
ALTER TABLE "Start" ALTER COLUMN "Zeit" TIMESTAMP(6)
```

Jetzt ist das Feld in der Lage, auch im Millisekundenbereich Zeiten aufzunehmen.

Das nächste Problem für ein Formular besteht darin, dass so ein Feld sowohl ein Datum als auch eine Zeit abspeichert. Deshalb erstellt Base auch im Formularassistenten oder bei dem Weg über das direkte Hinzufügen von Feldern für ein Datum/Zeit – Feld zwei Eingabemasken. Eine dient zur Eingabe des Datums, eine andere zur Eingabe der Zeit. Diese ist für eine einfache Zeitangabe mit Nachkommastellen nicht brauchbar. Stattdessen muss ein Feld her, dass beides darstellen könnte und außerdem in der Lage ist, auch noch die Hundertstel Sekunden mit aufzunehmen. Deshalb wird das Feld als «Formatiertes Feld» erstellt. Dieses Feld wird hier nur zur Anzeige benutzt und deswegen in **Eigenschaften → Allgemein → Nur Lesen** auf «Ja» gestellt. Zur Eingabe von Werten ist es so auf dem direkten Wege nicht geeignet, da neben der Zeiteingabe eine Datumseingabe erwartet wird, das Feld aber bei der vorliegenden Formatierung diese Datumseingabe nicht mitliefert.

Die Ermittlung der korrekten Zeit muss wieder über ein Makro geleistet werden.

Makro für das Formular «Einzelmessung_Zeitfeld»

```
SUB Zeitmessung_Timestamp
    DIM unoStmp AS NEW com.sun.star.util.DateTime
    DIM daStart AS NEW com.sun.star.util.DateTime
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oSubForm AS OBJECT
    DIM oSubFeld AS OBJECT
    DIM stZeit AS STRING
    DIM ar()
    DIM arMandS()
    DIM loNano AS LONG
    DIM inSecond AS INTEGER
    DIM inMinute AS INTEGER
    DIM inIndex AS INTEGER
    DIM lZeit AS LONG
```

```
    DIM lZeit1 AS LONG
```

Sämtliche Variablen werden zuerst deklariert. Hier ist darauf zu achten, dass die entsprechenden Zuweisungen für die Timestamp-Felder und für die Sekundenbruchteile, die Sekunden und Minuten stimmen. Die Zeitdefinition für die Sekundenbruchteile wird über das Struct von LibreOffice auf Nanosekunden festgelegt und benötigt daher eine Ganzzahl, die entsprechend groß genug ist, also vom Typ Long.

```
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.forms.getByName("Formular")
    oColumns = oForm.Columns
    inIndex = oForm.findColumn("Startzeit")
    daStart = oForm.getTimestamp(inIndex)
```

Das Formular wird angesteuert. Der Timestamp für den Start wird aus der dem Formular zugrundeliegenden Tabelle ausgelesen, damit aus diesem Feld der Datumsteil für das Feld «Zeit» ausgelesen werden kann, in dem schließlich ein kompletter Timestamp gespeichert werden muss.

Alternativ kann auch einfach das Standardstartdatum 1899-12-30 eingesetzt werden. Schließlich wird der Datumswert für die Zeitauswertung nicht benötigt.

```
    oFeld = oForm.getByName("Zeit")
    oSubForm = oForm.getByName("Zeitspeicher")
    oSubFeld = oSubForm.getByName("Hidden")
    lZeit1 = getSystemTicks()
    lZeit = oSubFeld.getCurrentValue
    doZeit = (Cdbl(lZeit1-lZeit))/1000
```

Die Zeit wird in Sekunden umgerechnet. Damit auch Nachkommastellen angezeigt werden, muss die Differenz der Felder **`lZeit1-lZeit`** mit **`Cdbl`** in eine Double-Variable umgewandelt werden. Die Tausendstel erscheinen jetzt hinter dem Dezimalpunkt. Die Zeitangabe wird in einen Text umgewandelt und am Dezimalpunkt aufgetrennt.

```
    ar() = Split(Str(doZeit),".")
    loNano = Clng(ar(1)&"000000")
```

Die Tausendstel-Sekunden werden aus dem 2. Teil des Arrays ausgelesen. Es müssen noch 6 Nullen angehängt werden, da Nanosekunden gespeichert werden.

```
    doZeit = Fix(doZeit)
    inMinute = Fix(doZeit/60)
    inSecond = doZeit - inMinute*60
```

Mit der Funktion Fix werden die Nachkommastellen der Zeitvariablen abgeschnitten. Für die Minutenangabe werden die verbleibenden Sekunden durch 60 dividiert. Die Sekunden werden ermittelt, indem der Rest aus der vorhergehenden Division bestimmt wird.

```
    WITH unoStmp
        .NanoSeconds = loNano
        .Seconds = inSecond
        .Minutes = inMinute
        .Hours = 0
        .Day = daStart.Day
        .Month = daStart.Month
        .Year = daStart.Year
    END WITH
    oFeld.BoundField.updateTimestamp(unoStmp)
    oForm.UpdateRow()
END SUB
```

Die einzelnen Variablen der Timestamps werden gesetzt. In manchen Anleitungen steht hierzu noch statt des Anteils **`NanoSeconds`** der Anteil **`HundrethSeconds`**. Für LibreOffice in der aktuellen Fassung ist diese Angabe nicht mehr korrekt.

Zum Schluss wird der Zeitstempel in das Anzeigefeld für die Zeit geschrieben und der Datensatz mit dem neuen Wert überschrieben.

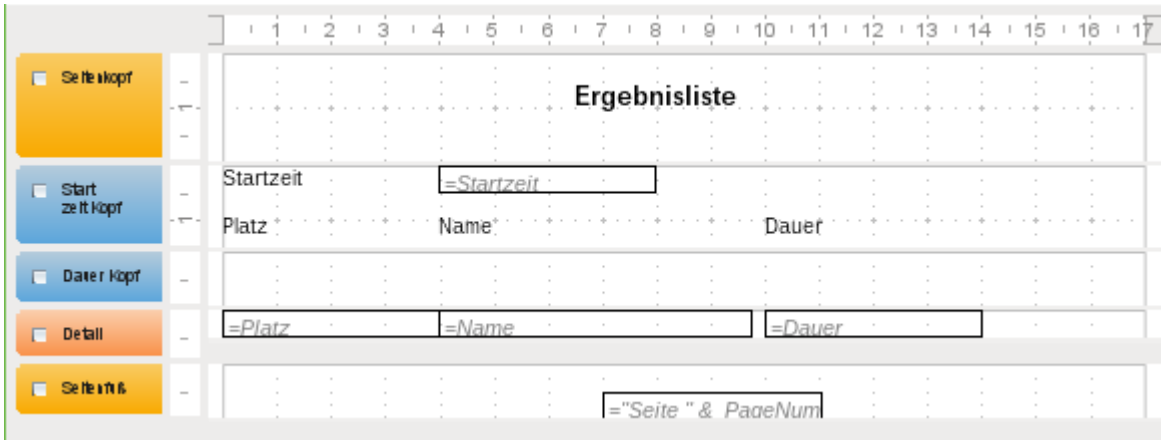
Berichte

Ergebnisliste

Startzeit	01.05.14 10:00	
Platz	Name	Dauer
1	Nils Weglauf	282958 ms
2	Uwe Maier	286620 ms
3	Xavers Franz	290068 ms
4	Karl Müller	296094 ms

Eine direkte Auswertung der Abfrage "Abfrage_Ziel" stellt die Ergebnisliste dar. Die Felder sind bereits in der Abfrage errechnet, so dass nur noch eine Gruppierung nach der Startzeit und nach der Platzierung notwendig ist.

Im Entwurf sieht dieser Bericht so aus:



Damit die Bezeichnungen «Platz», «Name» und «Dauer» nur pro Rennen einmal erscheinen sind sie in dem Gruppenkopf für die Startzeit verankert worden.

Der Gruppenkopf für das Feld «Dauer» dient nur der Sortierung nach der Zeit. Hier könnte also genauso gut das Feld «Platz» zur Sortierung genutzt werden. Dieser Gruppenkopf wird in dem Bericht nicht angezeigt (**Eigenschaften Gruppenkopf → Allgemein → Sichtbar: Nein**)

Auf jeder Seite erscheint eine Seitennummerierung in der Form Seite … von … .

Urkunde

Nils Weglauf

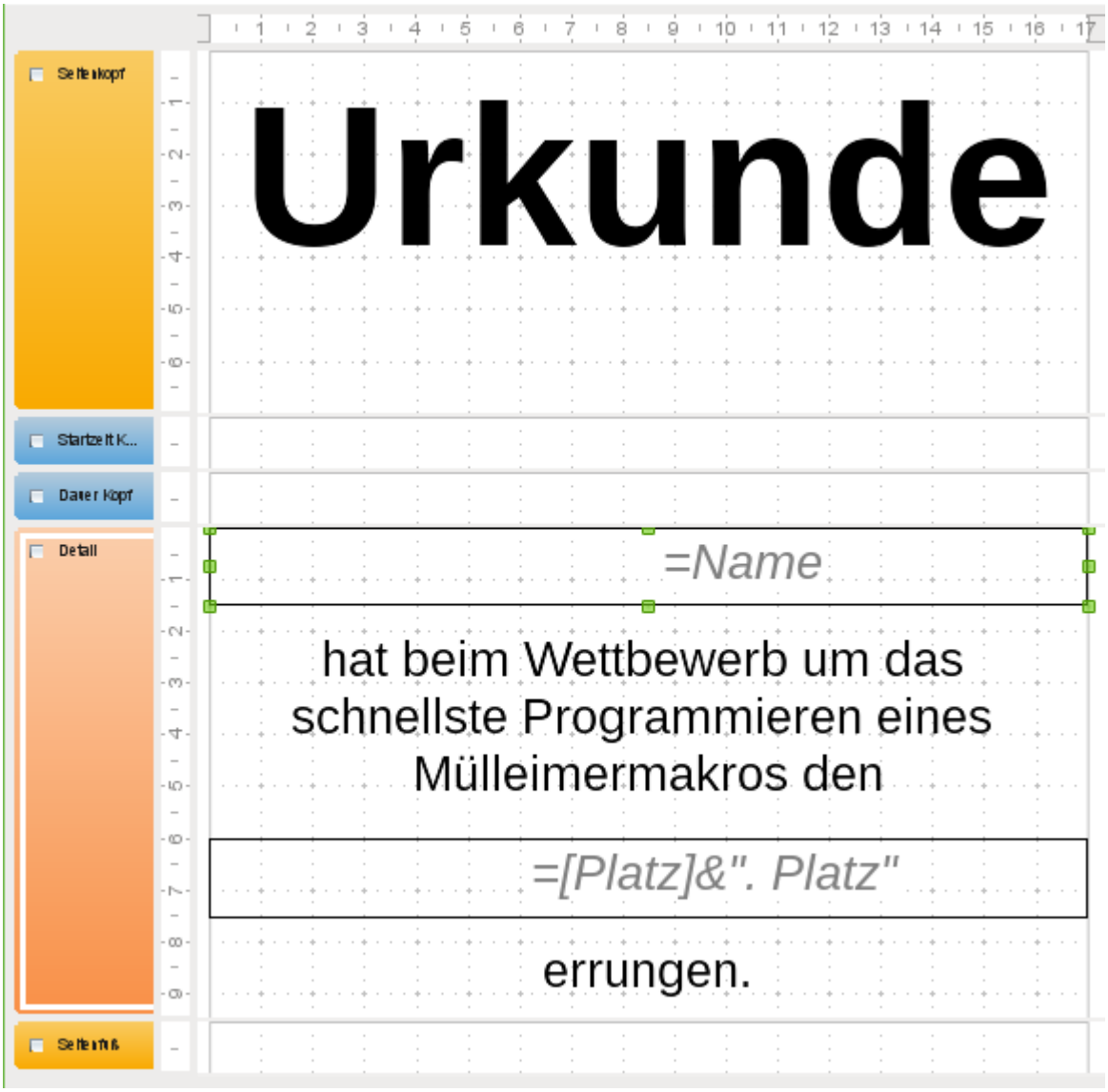
hat beim Wettbewerb um das
schnellste Programmieren eines
Mülleimermakros den

1. Platz

errungen.

Auch ein Urkundendruck ist problemlos möglich. Sinnvoll ist hier allerdings, den Druckjob möglichst zu vereinfachen. Die einfachste Lösung wäre, nur die veränderlichen Felder im Bericht zu positionieren und auf einen schön gestalteten Urkundenvordruck zu drucken. Je einfacher der Bericht gehalten wird desto schneller baut sich schließlich auch das Fenster für den Druck auf. Und je weniger grafische Elemente eingefügt sind, desto kürzer ist auch die Druckzeit, die während irgendeiner Veranstaltung sowieso recht knapp ist.

Die Blattübersicht ist hier stark verkleinert. Schließlich ist der Druck «Urkunde» im Original auf eine Schriftart von 110 Punkten gesetzt worden.



In der verkleinerten Ansicht des Editors erscheint das Feld «Name» nicht richtig zentriert. Der Ausdruck zeigt aber eine entsprechenden saubere Zentrierung.

Für die Ausgabe der Platzierung ist eine kleine Formel erforderlich, da sonst die Zentrierung Probleme bereitet. Mit [Platz] wird das entsprechende Feld aus der Datenquelle angesprochen, mit &". Platz" wird an die Platzierung aus dem Datenfeld ein Punkt, ein Leerzeichen und der Begriff «Platz» angefügt.

Damit nur eine Urkunde pro Seite gedruckt wird ist bei **Eigenschaften Detail → Allgemein → Seitenumbruch erzwingen: Nach Bereich** ausgewählt.

Serienbriefe direkt aus Base heraus

Für Serienbriefe wird in der Regel zuerst einmal eine Writer-Datei gestartet und dann von dort aus der Seriendruck in die Wege geleitet. Die folgende Variante soll aufzeigen, wie so etwas auf verschiedene Art auch direkt aus einem Base-Formular heraus erfolgen kann.

Tabellen

Die Tabellen haben in dieser Datenbank nur die Funktion, für eine Bestückung der Serienbriefe eine Grundlage zu bieten. Sie werden deshalb nur wenig zusätzlich erklärt.

In der Tabelle "Anschrift" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Vorname	Text	
Nachname	Text	
Geschlecht	Text	Länge des Textes ist 1 Zeichen. Hier wird nur 'm' oder 'w' eingetragen, damit später daraus eine Anrede ermittelt werden kann.
Straße_Nr	Text	
Postleitzahl	Text	Länge des Textes ist 5 Zeichen. Postleitzahlen sollten nicht als Zahlen abgespeichert werden. Führende Nullen sind sonst nicht möglich.
Ort	Text	

Die Tabelle "Anschrift" zeigt das typische Anwendungsbeispiel eines Serienbriefes. Hier werden beispielhaft lediglich die Adressdaten in ein Druckdokument übertragen, das natürlich mit entsprechenden Feldern beliebig erweitert werden könnte.

Die weiteren Tabellen dienen dazu, auch den Druck einer Rechnung direkt aus der Datenbank heraus aufzuzeigen. Eine Rechnung unterscheidet sich in sofern von einem einfachen Serienbrief, da hier zu einem Datensatz (z.B. der Anschrift) viele Rechnungstitel gehören. Im Report-Builder wird so etwas durch Gruppierungen gelöst. Hier ist der eingeschlagene Weg etwas anders angefasst worden.

In der Tabelle "Rechnung" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt. Der Primärschlüssel ist gleichzeitig die spätere Rechnungsnummer.
Datum	Datum	Hier wird das Rechnungsdatum festgelegt.
Anschrift_ID	Integer	Hier wird die "ID" der Tabelle "Anschrift" als Fremdschlüssel gespeichert. Grundsätzlich kann so eine Person mehrere Rechnungen erhalten ohne jedes Mal die gesamte Anschrift erneuern zu müssen.

Die Tabelle Rechnungsinhalt wird nur für die Durchführung des Rechnungsdruckes benötigt. Sie wird durch ein Makro gefüllt und nicht irgendwie durch Formulare bearbeitet.

In der Tabelle "Rechnungsinhalt" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Die ID ist identisch mit der der Tabelle "Rechnung". Eine Beziehung wird hier aber nicht festgelegt.
Rechnungsin-	Text	Länge des Textes sollte groß gewählt werden, da hier für eine

halt		der Serienbrieffunktionen alle Rechnungsinhalte in einem Datensatz gespeichert werden. Im Beispiel ist der Länge auf 10000 Zeichen festgelegt.

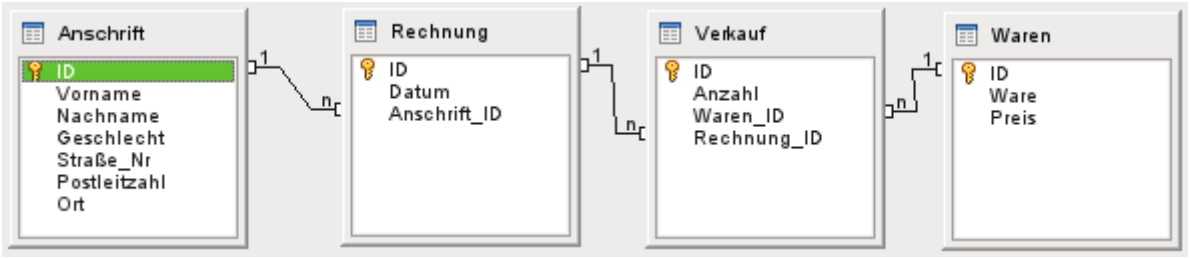
In der Tabelle "Verkauf" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Anzahl	Tiny Integer	Je größer die Anzahl der Waren sein sollte, desto größer ist natürlich der Umfang dieses Feldes festzulegen. Mit Tiny Integer geht die Anzahl von -128 bis +127 – für ein einfaches Beispiel völlig ausreichend.
Waren_ID	Integer	Hier wird die "ID" der Tabelle "Waren" als Fremdschlüssel gespeichert.
Rechnung_ID	Integer	Hier wird die "ID" der Tabelle "Rechnung" als Fremdschlüssel gespeichert.

In der Tabelle "Waren" werden die folgenden Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle. Sie ist als Autowert festgelegt.
Ware	Text	
Preis	Dezimal	Bei der Preisangabe ist darauf zu achten, dass 2 Nachkommastellen vorgesehen sind. Sonst sind nur ganzzahlige Angaben möglich. Das Feld kann auch entsprechend schon mit der Währungsangabe vorformatiert werden. Für die letztlich zu erfolgende Darstellung im Druck hat dies aber keine Bedeutung.

Die Tabellen sind für die Rechnungserstellung folgendermaßen verbunden:



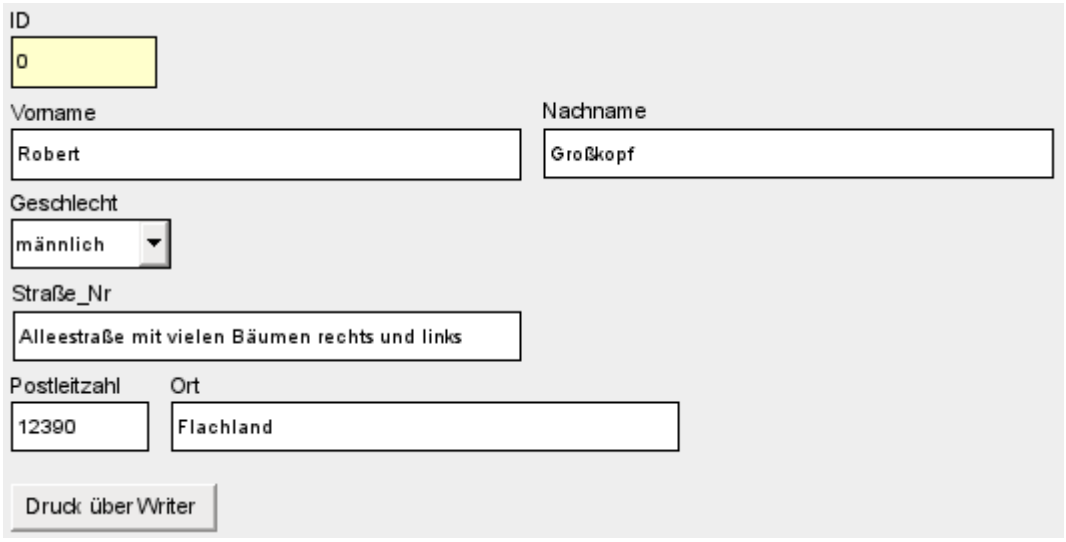
Die Tabelle Anschrift wird separat bestückt und dient als erstes Beispiel für den Seriendruck. Die Kombination mit der Rechnungserstellung zeigt dann ein erweitertes Anwendungsbeispiel.

Formulare

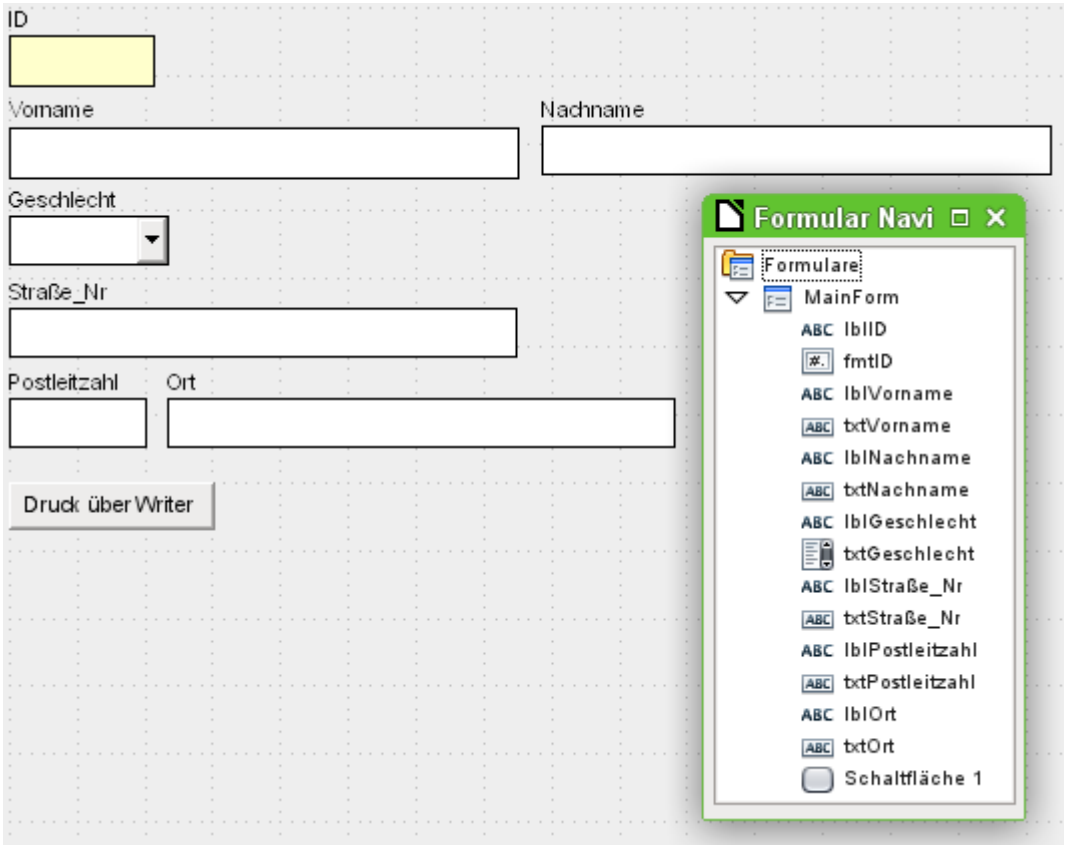
Die Formulare greifen mit unterschiedlichem Schwierigkeitsgrad auf die Serienbrieffunktion bzw. auf Textfelder eine Vorlage zu. Der einfachste Zugriff erfolgt für den Inhalt, der in der Tabelle "Anschrift" gespeichert wird.

Das Formulare «Anschrift»

Das erste Formular stellt ein einfaches Eingabeformular für Adressen dar.



Das Formular speichert die Daten in der Tabelle "Anschrift" ab. Das Primärschlüsselfeld ist farblich abweichend gekennzeichnet, da der Wert automatisch erstellt wird. Das Feld "Geschlecht" wird über ein Listenfeld beschrieben. Der Druck des aktuellen Datensatzes kann direkt über den Writer erfolgen. Über den Button wird hierzu ein Makro ausgeführt. Mehr dazu etwas weiter unten.



Der Formularnavigator zeigt keine Besonderheiten auf. Bei einem Formular, das lediglich zum Bestücken einer Tabelle genutzt wird, ist dies auch nicht weiter verwunderlich.

Das Listenfeld wird nicht über eine zusätzliche Tabelle per SQL geladen, sondern ist hier direkt über eine Werteliste definiert. Unter **Eigenschaften → Daten → Listeninhalt** sind die Werte «m» und «w». Um die Werte in dieser Form einzugeben muss lediglich das vorgesehene Feld mit dem Cursor betreten werden. Das Feld klappt auf und über die Eingabe von *Strg*+*Enter* kann nach der Eingabe des ersten Wertes in die zweite Zeile für einen zweiten Wert gewechselt werden.

Der Listeninhalt wird in die dem Formular zugrundeliegenden Tabelle "Anschrift" übertragen. Dargestellt werden allerdings die unter **Eigenschaften → Daten → Listen-Einträge** gemachten Einträge. Hier ist auf die selbe Reihenfolge zu achten, damit nicht bei der Anzeige «weiblich» in der Tabelle «m» abgespeichert wird.

Ausdruck aus dem Formular «Anschrift»

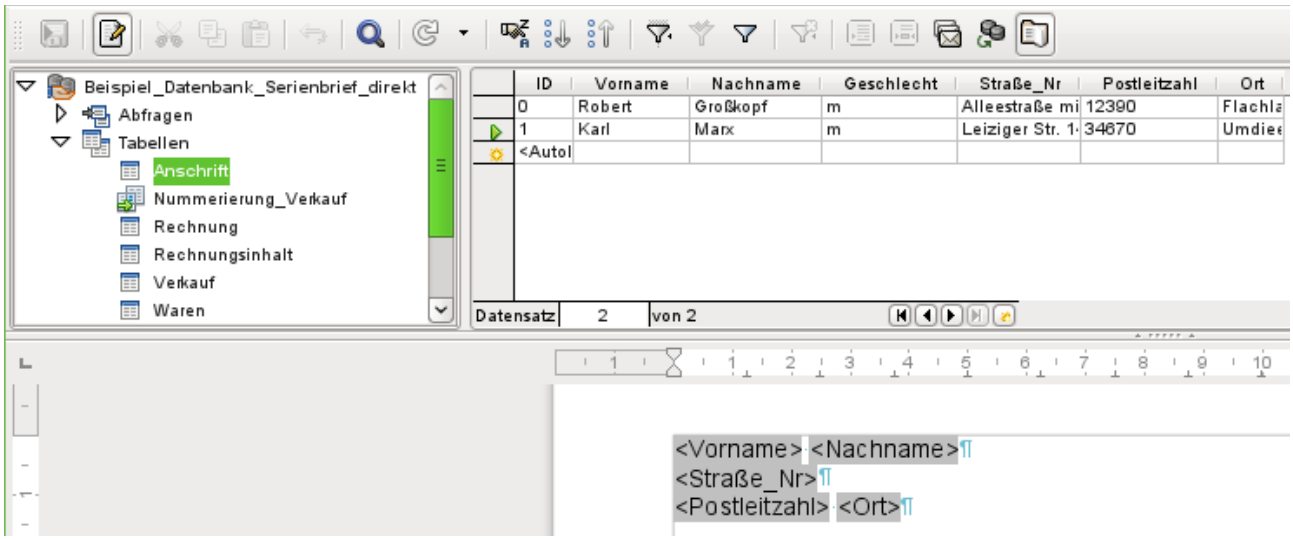
Bevor ein Ausdruck eines Serienbriefes über die Datei «Beispiel_Serienbrief.odt» erfolgen kann, muss die Datenbank in LibreOffice angemeldet werden:

Extras → Optionen → LibreOffice Base → Datenbanken → Neu

Damit das Makro richtig funktioniert, muss für die Datenbank der im Makro vermerkte Name «Beispiel_Datenbank_Serienbrief_direkt» gewählt werden. Datenbank und Textdokument für den Serienbrief müssen außerdem in dem gleichen Verzeichnis liegen.

Wird anschließend in einem Datensatz der Button Druck über Writer betätigt, so wird eine neue Textdatei mit dem im Serienbrief abgelegten Feldinhalt erstellt. Der Dateiname wird aus dem Nachnamen und einer folgenden 0 sowie der Endung «odt» zusammengefügt. Die Datei hat den selben Inhalt, wie ihn auch ein Druck, ausgelöst von der ursprünglichen Serienbriefdatei, hätte.

Der Serienbrief selbst wurde hier einfach über den Datenbankbrowser erstellt:



Mit F4 oder über Ansicht → Datenquelle oder auch über den entsprechenden Button in der Symbolleiste werden die Datenquellen angezeigt. Aus der Datenquelle wird die entsprechende Tabelle (hier «Anschrift») ausgesucht. Die Tabelle wird rechts davon angezeigt. Jetzt können die Serienbrieffelder durch einen Druck mit der linken Maustaste auf den jeweiligen Tabellenkopf in das darunterliegende Writer-Dokument gezogen werden.

Die Feldauswahl der Serienbrieffelder kann alternativ auch über **Einfügen → Feldbefehl → Andere → Datenbank → Seriendruckfeld** erfolgen.

Statt aber diesen Vorlagebrief zuerst zu starten, dann den Datensatz auszusuchen, den Druck zu starten und die Bedingungen dafür festzulegen, erfolgt die Erstellung des Zieldokumentes bei Betätigung von Druck über Writer direkt, ohne das Ausgangsdokument auf dem Bildschirm zu öffnen.

Dieser Zugriff auf das Dokument wird über das Makro «Serienbrief» erreicht.

Makrosteuerung zum Ausdruck im Serienbrief

```
SUB Serienbrief
    DIM oForm AS OBJECT
    DIM oDB AS OBJECT
    DIM stDir AS STRING
    DIM loFeldID AS LONG
    DIM loID AS LONG
```

Der Datenbankwertebereich für "Integer" entspricht dem StarBasic-Wertebereich für "Long". Deshalb müssen Integer-Felder aus der Datenberg mit einer Long-Variablen ausgelesen werden, wenn wirklich alle Werte erfasst werden sollen.

```
    DIM arProps()
```

Die nötigen Variablen für den Serienbrief werden in dem Array **`arProps()`** gespeichert.

Nach der Deklaration aller Variablen wird aus dem aktuell im Formular angezeigten Datensatz der Primärschlüsselwert ausgelesen. Dadurch wird erreicht, dass tatsächlich nur ein Datensatz anschließend für den Druck weitergegeben wird. Es wird davon ausgegangen, dass das Feld in der Tabelle die Bezeichnung "ID" hat und die Tabelle aktuell im Formular «MainForm» als Datengrundlage dient. Das Feld "ID" muss für dieses Vorgehen nicht im Formular als Formularfeld angezeigt werden. Der Zugriff geht hier direkt über die in der Datengrundlage des Formular vorhandenen Felder.

```
    oForm = thisComponent.Drawpage.Forms.MainForm
    loFeldID = oForm.findColumn("ID")
    loID = oForm.getLong(loFeldID)
```

Die Variable «loID» enthält jetzt den aktuellen Primärschlüsselwert.

```
    MailMerge = CreateUnoService("com.sun.star.text.MailMerge")
    oDB = ThisComponent.Parent
```

Der Zugriff auf die URL ist nicht vom Formular aus direkt möglich. Es muss auf den darüber liegenden Frame der Datenbank Bezug genommen werden, damit der Pfad ermittelt werden kann, in dem die Datenbank zur Zeit liegt.

```
    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
```

Der Titel der Datenbank wird von der URL abgetrennt. Anschließend werden die verschiedenen Details für den UnoService festgelegt. Dies kann entweder über eine jeweils einzelne Definition wie **`MailMerge.DataSoureName`**, **`MailMerge.DocumentURL`** usw. **oder** über die Anweisung **`WITH MailMerge`** und der Aufzählung der Detailanweisungen erfolgen.

```
    WITH MailMerge
        .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
```

Die Datenbank muss unter diesem Namen eingebunden sein.

```
        .DocumentURL = stDir+"Beispiel_Serienbrief.odt"
```

Die Datei «Beispiel_Serienbrief.odt» liegt im gleichem Pfad wie die Datenbank.

```
            .SaveAsSingleFile = true
```

Aus dem Inhalt sollen Einzeldokumente erstellt werden, nicht mehrere Briefe hintereinander in einem Dokument. **`false`** oder **`0`** bedeutet «nein», **`true`** oder **`1`** bedeutet «ja».

```
            .CommandType = 0
```

Die folgenden Datenquellentypen gibt es: 0 = Tabelle, 1 = Abfrage, 2 = SQL-Code

```
            .Command = "Anschrift"
```

Abhängig von dem Typen wird hier entweder die Bezeichnung der Tabelle/Abfrage oder der komplette SQL-Code angegeben.

```
            .Filter = "ID ="+loID
```

Der Wert des Primärschlüssels wird hier als Filter für den SQL-Code an den Serienbrief übergeben.

```
            .FileNameFromColumn = true
```

Hier wird angegeben, dass der Bezeichner für den Brief aus dem Inhalt eines Tabellenfeldes erstellt werden soll

```
            .FileNamePrefix = "Nachname"
```

Aus dem Inhalt dieses Tabellenfeldes wird der Bezeichner für die Datei übernommen. Dabei wird hinter den Nachnamen eine **`0`** gesetzt, um gegebenenfalls mehrere Serienbriefe an eine Person mit gleichem Namen zu senden. Der folgende Brief erhält dann eine **`1`** usw.

```
            .OutputType = 2
```

Der Serienbrief kann direkt an den Drucker gehen (Drucker = 1), er kann als Datei abgespeichert (Datei = 2) oder als E-Mail versandt werden (Mail = 3).

```
            .OutputUrl = stDir
```

Die zu erstellende Datei wird hier im selben Pfad abgelegt, in dem sich auch die Datenbank befindet. Hier könnte natürlich jeder beliebige Pfad eingegeben werden.

```
        END WITH
        MailMerge.execute(arProps())
    END SUB
```

Das Formular «Anschrift_Textfelder»

Vom Aufbau her unterscheidet sich das Formular «Anschrift_Textfelder» nicht vom Formular «Anschrift». Allerdings ist in den Formular-Eigenschaften unter Daten → Art des Dateninhaltes «Abfrage» ausgewählt. Dies ist nur notwendig, da in den Adressen eine Anrede dargestellt werden soll.

Die Abfrage, auf der das Formular beruht, zeigt die gesamte Tabelle "Anschrift" auf:

```
SELECT
    "ID", "Vorname", "Nachname", "Geschlecht" AS "Geschl_kurz",
    "Straße_Nr", "Postleitzahl", "Ort",
    CASEWHEN( "Geschlecht" = 'm', 'Herrn', 'Frau' ) AS "Geschlecht"
    FROM "Anschrift"
```

Das ursprüngliche Feld "Geschlecht" wird mit einem Alias versehen, da die selbe Bezeichnung jetzt für die Anrede genutzt werden soll. Mit der Funktion **`CASEWHEN`** wird ausgelesen, ob in dem Feld "Geschlecht" der Wert «m» steht. Steht dort dieser Wert, so wird «Herrn» ausgegeben. Ansonsten erscheint «Frau».

Ausdruck aus dem Formular «Anschrift_Textfelder»

Das Formular «Anschrift_Textfelder» arbeitet nicht mit der Serienbrieffunktion von LibreOffice zusammen. Stattdessen wird auf Platzhalter zugegriffen.

Zuerst muss im Writer eine Vorlage mit Platzhaltern erstellt werden. Dies geschieht über **Einfügen → Feldbefehl → Funktionen → Platzhalter**. Die Platzhalter werden der Einfachheit halber so benannt, wie das entsprechende Feld in der Tabelle bzw. in der Abfrage heißt, deren Inhalt der Platzhalter annehmen soll.

Für einfache Zwecke reicht hier der Typ «Text», mit den anderen Varianten, z.B. «Grafik», können dann weitere Inhalte umgesetzt werden.

In dem Makro wird der Pfad zur Vorlage hinterlegt. Von dem Makro werden die Platzhalter befüllt. Es wird ein neues Dokument erstellt, das direkt mit den Inhalten gefüllt wird. Das damit geöffnete Dokument muss nur noch abgespeichert oder direkt ausgedruckt werden.

Makrosteuerung zum Ausdruck mit Textfeldern

```
SUB Textfelder_Fuellen
    DIM oForm AS OBJECT
    DIM oColumns AS OBJECT
    DIM oDB AS OBJECT
    DIM oNewDoc AS OBJECT
    DIM oTextfields AS OBJECT
    DIM oTextfield AS OBJECT
    DIM stColumnname AS STRING
    DIM stDir AS STRING
    DIM inIndex AS INTEGER
    oForm = thisComponent.Drawpage.Forms.MainForm
```

Nach der Definition der Variablen wird das Hauptformular angesteuert. Hier könnte auch die Lage des auslösenden Buttons das Formular selbst ermitteln.

```
    oColumns = oForm.Columns
    oDB = ThisComponent.Parent
    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
```

Der Zugriff auf die URL wird wie im Makro «Serienbrief» geregelt. Die Vorlage «Beispiel_Textfelder.ott» liegt im gleichen Pfad wie die Datenbank.

```
    stDir = stDir & "Beispiel_Textfelder.ott"
```

Die Vorlage wird geladen.

```
DIM args(0) AS NEW com.sun.star.beans.PropertyValue
args(0).Name = "AsTemplate"
args(0).Value = True
oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Textfelder aus dem Dokument werden ausgelesen. Zu beachten ist, dass hier für die Gesamtzahl der Felder die Variable **`oTextfields`** (mit einem «s»), für das einzelne Textfeld **`oTextfield`** gewählt wurde.

```
oTextfields = oNewDoc.Textfields.createEnumeration
DO WHILE oTextfields.hasMoreElements
   oTextfield = oTextfields.nextElement
   IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
      stColumnname = oTextfield.PlaceHolder
```

`Placeholder` ist die Benennung für das Textfeld in dem Writer-Dokument.

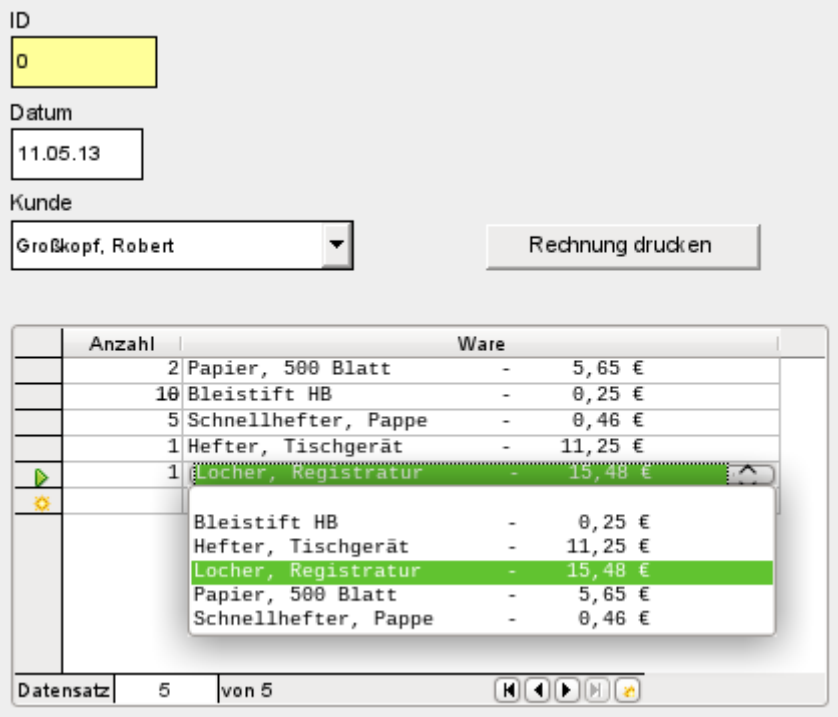
```
         IF oColumns.hasByName(stColumnname) THEN
```

Wenn der Name des Textfeldes gleich dem Spaltennamen der Daten ist, die dem Formular zugrunde liegen, dann wird dem Textfeld der Inhalt dieses Feldes zugewiesen. Da für das Formular eine Abfrage als Datenbasis gewählt wurde, wird jetzt aus dieser z.B. der Wert für das Feld "Geschlecht" ermittelt. Das ist dank der Abfrage aber mit den Inhalten der zum Geschlecht passenden Anrede gefüllt.

```
            inIndex = oForm.findColumn(stColumnname)
            oTextfield.Anchor.String = oForm.getString(inIndex)
         END IF
      END IF
   LOOP
END SUB
```

Das Formular «Rechnung»

Beim Formular «Rechnung» werden einige zusätzliche Problemstellungen angegangen. Die erste Problemstellung zeigt bereits der Screenshot während einer Eingabe: Im Listenfeld sind neben den Namen die Preis angezeigt – allerdings gleich so, dass das ganze wie eine Tabelle aussieht.



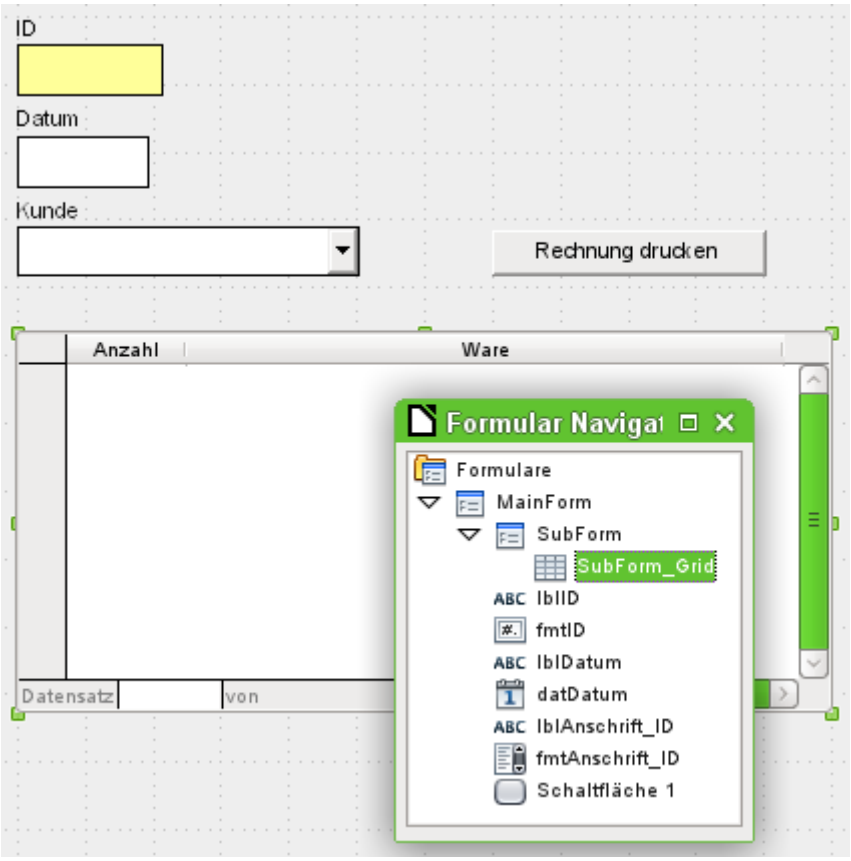
Um ein **Listenfeld** so wie oben erscheinen zu lassen muss zuerst einmal die Schriftart von einer proportionalen Schriftart zu einer Schriftart mit fester Zeichenbreite umgewandelt werden. Das Tabellenkontrollfeld wird markiert. Über das **Kontextmenü → Kontrollfeld → Eigenschaften: Tabellen-Steuerelement → Allgemein → Schrift** wird die Schriftart «Liberation Mono, Standard» ausgesucht.

Anschließend wird der Tabellenkopf «Ware» angeklickt. Über das **Kontextmenü → Spalte → Eigenschaften: Listenfeld → Daten → Art des Listeninhalts** wird «SQL» gewählt. Danach wird der **Listeninhalt** festgelegt:

```
SELECT
   LEFT("Ware"|| SPACE(25), 25) || ' - ' ||
      REPLACE( RIGHT( SPACE(8) ||"Preis", 8),'.',',') || ' €',
      "ID"
   FROM "Waren"
   ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

Es werden Daten aus der Tabelle "Waren" ausgelesen. Mit **||** werden Elemente verschiedener Felder bzw. auch Text verbunden. An den Inhalt des Feldes werden 25 Leerzeichen (**SPACE**) angehängt. Anschließend wird der so entstandene Text von links aus (**LEFT**) auf 25 Zeichen zurechtgestutzt. Dadurch ist die Zeichenlänge für die Darstellung der Ware festgelegt. Der "Preis" wird ebenfalls mit einer entsprechenden Anzahl an Leerzeichen gekoppelt. Da der Preis aber rechtsbündig dargestellt wird, werden jetzt 8 Zeichen von rechts aus (**RIGHT**) gelesen. Höhere Preise sind in der Datenbank sowieso nicht vorgesehen. Damit der Preis außerdem mit einem Dezimalkomma und nicht mit einem Dezimalpunkt dargestellt wird, wird über **REPLACE** der Punkt durch ein Komma ersetzt. Schließlich wird noch das €-Zeichen an den Gesamtstring angehängt.

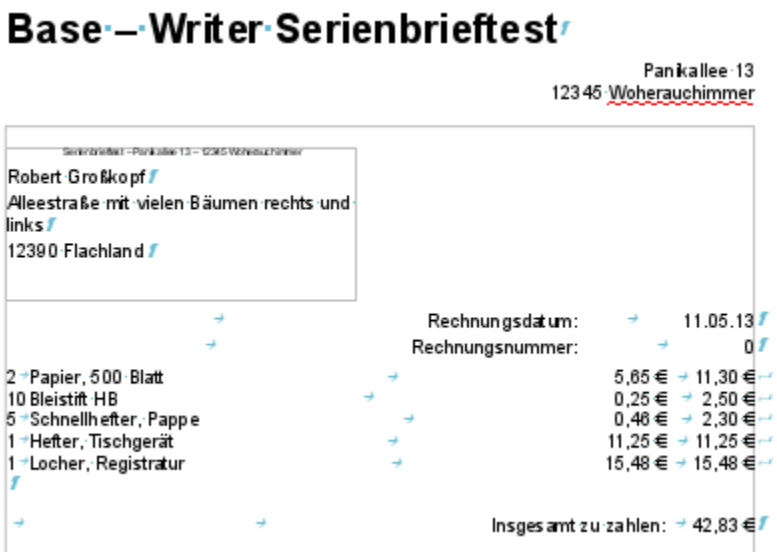
Wie die Anzahl der erforderlichen Leerzeichen auch automatisch ermittelt werden kann ist im Base-Handbuch beschrieben.



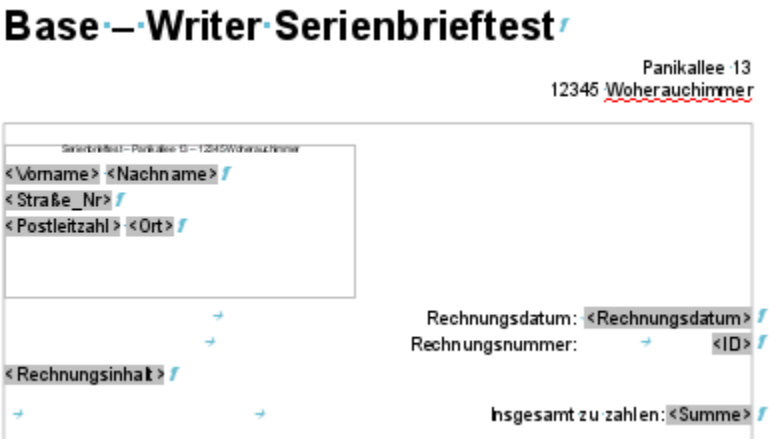
Der Blick auf den Formularnavigator zeigt, dass in diesem Formular neben dem Hauptformular ein Unterformular existiert. Die Datenquelle des Hauptformulars ist die Tabelle "Rechnung", das Unterformular hat als Datenquelle die Tabelle "Verkauf". Die Verbindung erfolgt von "Rechnung"."ID" nach "Verkauf"."Rechnung_ID". So ein Formular mit Unterformular wird auch vom Formularassistenten direkt erstellt, wenn vorher die Tabellenbeziehung über **Extras → Beziehungen** definiert wurde.

Ausdruck aus dem Formular «Rechnung»

Wird über Rechnung drucken der Seriendruck gestartet, so läuft der Prozess genau wie beim Seriendruck aus dem Formular «Anschrift» im Hintergrund ab. Es wird eine Datei erstellt, die wie im folgenden Bild zu sehen gefüllt ist:



Durch die seit LO 4.3 farbige Anzeige der Steuerzeichen wird im Ausdruck deutlich, wie eine Formatierung in Tabellenform erreicht wurde. Der Rechnungsinhalt wurde durch Tabulatoren und weiche Umbrüche in das Dokument eingefügt. Die Lage der Tabulatoren im Absatz ist über das Absatzformat der Vorlage definiert. Sie kann natürlich noch angepasst werden, falls die Preise oder auch die Anzahl mehr Platz benötigen. Die Länge des Gesamtinhaltes ist allerdings durch die Zeile begrenzt.



In der Vorlage erscheint für den Rechnungsinhalt nur das Serienbrieffeld **`< Rechnungsinhalt >`**. Es wird, wie alle anderen Felder, aus einem einzigen Datensatz der Datenquelle ausgelesen. Der wesentliche Unterschied zur Erstellung eines Seriendrucks für die Anschrift liegt in der Erstellung des Inhaltes für das Feld "Rechnungsinhalt".

Makrosteuerung zum Ausdruck der Rechnung im Serienbrief

Das Makro, das die Inhalte aus der Datenbank an den Serienbrief für die Rechnung weiterleiten soll, unterscheidet sich nur wenig von dem, das genau das gleiche Vorgehen für die Anschrift vorsieht. Im folgenden sind deshalb nur die Unterschiede der entsprechenden Prozedur aufgezeigt.

```
SUB Rechnung
    ...
    Rechnungsinhalt_zusammenstellen(loID)
```

Es wird von der Prozedur «Rechnung» aus eine andere Prozedur «Rechnungsinhalt_zusammenstellen» mit dem Primärschlüsselwert des aktuellen Datensatzes aufgerufen.

```
    WITH MailMerge
        .DataSourceName = "Beispiel_Datenbank_Serienbrief_direkt"
        .DocumentURL =  stDir+"Beispiel_Rechnung.odt"
        .Command = "Rechnung_einzeilig"
        ...
```

Der Aufruf von **`MailMerge`** unterscheidet sich nur in den Variablen, die das zu benutzenden Dokument und die in dem Dokument zu findende Abfrage beschreiben. Die Datenbank selbst ist oben nur der Vollständigkeit halber noch einmal aufgeführt. Die weiteren Variablen sind identisch.

```
    END WITH
    MailMerge.execute(arProps())
END SUB
```

Die HSQLDB bietet keine Funktion wie GroupConcat (MySQL) oder List (Firebird). Inhalte von gleichen Feldern in unterschiedlichen Datensätzen können deshalb am besten mit einem Makro in einer Zeile zusammengeführt werden. Das Ergebnis dieses Makros ist also das Erstellen eines einzigen Datensatzes aus der Abfrage beliebig Datensätze einer Tabelle.

```
SUB Rechnungsinhalt_zusammenstellen(loID AS LONG)
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oAbfrageergebnis AS OBJECT
    DIM stSql AS STRING
    DIM stText AS STRING
    stText = ""
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
```

Die Variablen werden deklariert. Die Variable **`loID`** wird beim Start der Prozedur direkt mitgegeben. Anschließend wird eine Verbindung zur Datenbank auf SQL-Ebene hergestellt. Der folgende SQL-Befehl ist dabei auf mehrere Zeilen verteilt, hier aber zur etwas besseren Lesbarkeit in eine Zeile zusammengefasst und gegliedert worden. Die Feldnamen der angesprochenen Tabelle müssen dabei in StarBasic mit doppelten Anführungszeichen eingegeben werden. Das doppelte Anführungszeichen bewirkt, dass das erste Anführungszeichen als Maskierung für das folgende sorgt und so ein Anführungszeichen tatsächlich aus dem Code heraus weiter gegeben wird.

```
    stSql = "SELECT ""Verkauf"".""Anzahl"" || CHAR( 9 ) ||
          ""Waren"".""Ware"" || CHAR( 9 ) ||
          REPLACE( ""Waren"".""Preis"" || ' €', '.', ',' ) || CHAR( 9 ) ||
          REPLACE( ""Verkauf"".""Anzahl"" * ""Preis"" || ' €', '.', ',' ) ||
              CHAR( 10 )
       FROM ""Verkauf"", ""Waren""
       WHERE ""Verkauf"".""Waren_ID"" = ""Waren"".""ID""
          AND ""Verkauf"".""Rechnung_ID"" = "+loID+""
```

Das Zeichen **`CHAR(9)`** steht für einen Tabulator. Das Zeichen **`CHAR(10)`** für den weichen Zeilenumbruch. Die Bedeutung der verschiedenen Steuerzeichen kann hier mit entsprechender Zahlenzuordnung nachgelesen werden: *http://de.wikipedia.org/wiki/Steuerzeichen* .

Die verschiedenen Felder werden wieder über **||** miteinander zu einem Feldinhalt verbunden. Bei den Feldern, die einen Preis ergeben sollen, wird der Dezimalpunkt, den sonst die Abfrage ergäbe, durch ein Dezimalkomma ersetzt. Außerdem wird die Währungseinheit «€» hinzugefügt.

Es werden nur die Felder der betroffenen Tabellen ausgewählt, die zu der entsprechenden Rechnungsnummer passen: **""Verkauf""."Rechnung_ID"" = "+loID+""**. Anschließend wird die Abfrage an die Datenbank gestellt.

```
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql) ' Ergebnis auswerten
    WHILE oAbfrageergebnis.next
       stText = stText + oAbfrageergebnis.getString(1)
    WEND ' nächster Datensatz
```

Jeder Datensatz des Abfrageergebnis wird ausgelesen und an den vorhergehenden Datensatz angehängt. Das Abfrageergebnis ist ein Text und steht an der ersten Position der Abfrage. Deswegen steht in der Anweisung **getString(1)**. Die Abfrage gibt nich mehr als ein Feld wieder.

```
    stSql = "DELETE FROM ""Rechnungsinhalt"" WHERE ""ID"" = "+loID+""
    oSQL_Anweisung.executeUpdate(stSql)
```

Die Tabelle "Rechnungsinhalt" wird von allen vorhergehenden Eingaben gelöscht. Anschließend werden die gerade zusammengestellten Daten als eine einzelne Zeile eingefügt.

```
    stSql = "INSERT INTO ""Rechnungsinhalt"" (""ID"",""Rechnungsinhalt"") VALUES
       ("+loID+", '"+stText+"')"
    oSQL_Anweisung.executeUpdate(stSql)
END SUB
```

Zusammenführen der Druckdaten in einer Abfrage

Nach Durchführung der Makros steckt der Inhalt für die Rechnung jetzt jeweils in einer Zeile der entsprechenden Tabellen. Jetzt müssen für den Serienbrief die Informationen aus den Tabellen zusammen gezogen werden. Außerdem ist noch der Gesamtbetrag der Rechnung durch Addition der Einzelbeträge zu ermitteln. Das alles leistet die Abfrage «Rechnung_einzeilig».

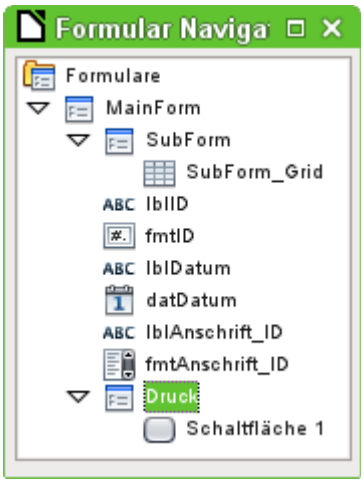
```
SELECT
   "Anschrift"."Vorname",
   "Anschrift"."Nachname",
   "Anschrift"."Straße_Nr",
   "Anschrift"."Postleitzahl",
   "Anschrift"."Ort",
   "a"."Datum" AS "Rechnungsdatum",
   "a"."ID",
   "Rechnungsinhalt"."Rechnungsinhalt",
   REPLACE( (
      SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" )
         FROM "Verkauf", "Waren"
         WHERE "Verkauf"."Waren_ID" = "Waren"."ID"
            AND "Verkauf"."Rechnung_ID" = "a"."ID"
      ) || ' €', '.', ',' ) AS "Summe"
   FROM "Rechnung" AS "a", "Anschrift", "Rechnungsinhalt"
   WHERE "a"."Anschrift_ID" = "Anschrift"."ID" AND "a"."ID" =
"Rechnungsinhalt"."ID"
```

Der Tabelle "Anschrift" werden alle notwendigen Daten für die Anschrift in dem Rechnungsformular entnommen. Die Tabelle "Rechnung" wird über einen Alias als "a" angesprochen. Dies ist notwendig, damit in der korrelierenden Unterabfrage nur die Beträge addiert werden, die zu der entsprechenden Rechnungsnummer (**"a"."ID"**) passen. Bei der Summenbildung **SUM("Verkauf"."Anzahl" * "Waren"."Preis")** wird auf die Rechnungsnummer des Datensatzes außerhalb der Unterabfrage so Bezug genommen. Auch in der Abfrage ist schließlich wieder das Ersetzen des Dezimalpunktes durch das Dezimalkomma anzutreffen.

Diese Abfrage wird mit dem Makro «Rechnung» als Datengrundlage für den Serienbrief «Beispiel_Rechnung.odt» genutzt.

Das Formular «Rechnung_Textfelder»

Das Formular «Rechnung_Textfelder» unterscheidet sich von dem Formular «Rechnung» nur in einem Punkt: Der Button liegt in einem weiteren Unterformular.



Das Unterformular «Druck» ist über das Feld "ID" mit dem Hauptformular verbunden. Art des Inhaltes ist eine Abfrage, und zwar die Abfrage «Rechnung_Textfelder».

Diese Abfrage stellt alle Einzeldaten der Rechnung zusammen, nicht aber den Rechnungsinhalt. Während die Einzeldaten anschließend in Textfelder übertragen werden, wird der Rechnungsinhalt in eine Writer-Tabelle eingetragen.

```
SELECT
   "Anschrift"."Vorname",
   "Anschrift"."Nachname",
   "Anschrift"."Straße_Nr",
   "Anschrift"."Postleitzahl",
   "Anschrift"."Ort",
   RIGHT( '0' || DAY( "a"."Datum" ), 2 ) || '.' || RIGHT( '0' ||
   MONTH( "a"."Datum" ), 2 ) || '.' ||         YEAR( "a"."Datum" ) AS
   "Rechnungsdatum",
   "a"."ID",
   REPLACE( (
      SELECT SUM( "Verkauf"."Anzahl" * "Waren"."Preis" )
         FROM "Verkauf", "Waren"
         WHERE "Verkauf"."Waren_ID" = "Waren"."ID"
            AND "Verkauf"."Rechnung_ID" = "a"."ID"
      ) || ' €', '.', ',' ) AS "Summe"
   FROM "Rechnung" AS "a", "Anschrift"
   WHERE "a"."Anschrift_ID" = "Anschrift"."ID"
```

Zuerst werden die Anschrifts-Daten erfasst.

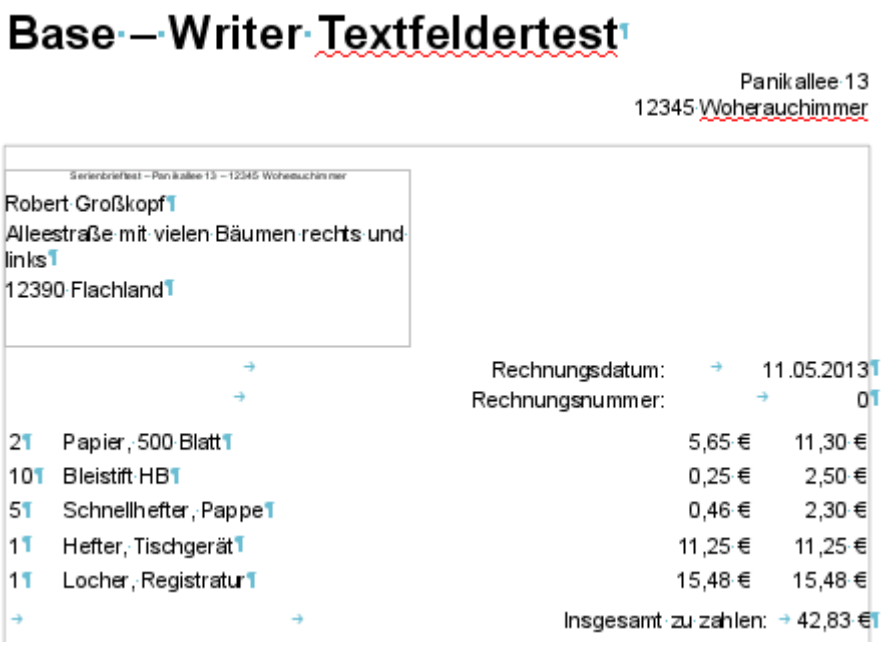
Beim Rechnungsdatum ist darauf zu achten, dass es, sofern in ein Textfeld eingelesen wird, in der korrekten Formatierung übergeben wird. Wird es nicht entsprechend vorformatiert, so erscheint es in der Schreibweise, die bei Datenbanken Standard ist: 2014-05-07. Mit **`DAY( "a"."Datum" )`** wird aus der Tabelle "Rechnung" (hier angesprochen über den Alias "a") der Tag des Datums ausgelesen – also z.B. «7». Der Tag soll allerdings zweistellig dargestellt werden, gegebenenfalls also mit einer führenden «0». Dies wird über **`RIGHT( '0' || DAY( "a"."Datum" ), 2 )`** schließlich erreicht. Es wird eine «0» vor die «7» gesetzt und anschließend von rechts aus die letzten

zwei Zeichen gelesen. Damit werden gleichzeitig zweistellige Tage, vor die eine «0» gesetzt wurde, wieder korrekt dargestellt. Entsprechend dem Tag wird auch mit dem Monat verfahren. Die Punkte als Trenner zwischen Tag und Monat sowie Monat und Jahr werden ebenfalls in die Verbindung über **| |** mit einbezogen.

Der weitere Inhalt der Abfrage wurde bereits für das vorhergehende Formular erläutert. Die Summe wird ermittelt, der Dezimalpunkt durch ein Komma ersetzt und schließlich noch ein «€»-Zeichen angefügt.

Ausdruck aus dem Formular «Rechnung_Textfelder»

Der Ausdruck aus dem Formular «Rechnung_Textfelder» ähnelt erst einmal sehr dem Ausdruck über den Serienbrief aus dem Formular «Rechnung».



Bei genauer Beobachtung vor allem der Steuerzeichen wird allerdings klar, dass hier für den Rechnungsinhalt eine andere Darstellung gewählt wurde. Hinter der Anzahl und den Warenbezeichnungen sind Absatzendmarken zu sehen.

Die Vorlage gibt näheren Aufschluss darüber, wie diese Absatzendmarken zustande kommen. Sie gehören zu den ersten zwei Spalten einer Tabelle. Die anderen Spalten sind rechtsbündig ausgerichtet und haben daher keine sichtbare Absatzendmarke. Die Tabelle ist hier nur zum besseren

Verständnis eingefärbt worden. Vorteil dieser Vorlage gegenüber der des Formulars «Rechnung» ist, dass der Inhalt in einer Tabellenzelle auch ruhig etwas größer sein kann. Das Layout wird nicht durcheinander kommen, wenn die Warenbeschreibung über mehrere Zeilen geht. Leider erfordert dieser Vorteil auch eine etwas erweiterte Anwendung der Makroprogrammierung.

Lediglich die in der Abfrage erstellten Felder werden auch als Felder in dem Ausdruck benötigt. Der restliche Inhalt wird über ein Makro direkt in die betreffenden Zellen geschrieben.

Makrosteuerung zum Ausdruck der Rechnung mit Textfeldern

```
SUB Rechnungsinhalt_zusammenstellen_Tabelle
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oAbfrageergebnis AS OBJECT
    DIM oForm AS OBJECT
    DIM oColumns AS OBJECT
    DIM oDB AS OBJECT
    DIM oNewDoc AS OBJECT
    DIM oTabelle AS OBJECT
    DIM oRows AS OBJECT
    DIM oTextfields AS OBJECT
    DIM oTextfield AS OBJECT
    DIM stColumnname AS STRING
    DIM stDir AS STRING
    DIM stSql AS STRING
    DIM stText AS STRING
    DIM inIndex AS INTEGER
    DIM i AS INTEGER
    DIM loID AS LONG
    DIM doPreis AS DOUBLE
    DIM doAnzahlPreis AS DOUBLE
    oForm = thisComponent.Drawpage.Forms.MainForm.getByName("Druck")
    oForm.reload()
    oForm.last()
```

Das Formular, in dem sich auch der Button befindet, wird angesprochen. Der Inhalt der Abfrage, die diesem Formular zugrunde liegt, soll an die Textfelder weitergegeben werden. Damit sicher alle Datensätze angezeigt werden, wird die Abfrage noch einmal mit **`reload()`** neu geladen.

Der Datensatzzeiger muss hier wegen eines Bugs in der 4.3.0.0 Beta1 auf den letzten Datensatz gesetzt werden, da beim **`reload()`** der Zeiger einen Datensatz weiter bewegt wird.

Die Befüllung der Textfelder ist gleich der in der Prozedur «Textfelder_Fuellen». Lediglich die Vorlage hat hier einen anderen Namen, nämlich «Beispiel_Rechnung_Textfelder.ott».

Nach dem Befüllen der Textfelder beginnt der Teil, der für den Druck der Rechnungsinhalte einen Gewinn bringt. Hier wird die Tabelle befüllt und mit neuen Zeilen versehen.

```
    oTabellen = oNewDoc.getTextTables
    oTabelle = oTabellen.getByName("Rechnungsinhalt")
```

Das aus der Vorlage geöffnete Dokument wird angesteuert. Die Tabelle ist in den Tabelleneigenschaften mit dem Namen «Rechnungsinhalt» versehen worden. So kann genau diese Tabelle aus dem Dokument herausgesucht werden.

```
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
```

Die Verbindung zur Datenbank wird erstellt. Anschließend wird über diese Verbindung ein SQL-Kommando abgesetzt, mit dem der Inhalt für die Tabelle «Rechnungsinhalt» zeilenweise gefüllt werden soll.

```
    oSQL_Anweisung = oVerbindung.createStatement()
```

```
stSql = "SELECT ""Verkauf"".""Anzahl"", ""Waren"".""Ware"", ""Waren"".""Preis"",
    ""Verkauf"".""Anzahl"" * ""Waren"".""Preis"" FROM ""Verkauf"", ""Waren"" WHERE
    ""Verkauf"".""Waren_ID"" = ""Waren"".""ID"" AND ""Verkauf"".""Rechnung_ID"" =
    "+loID+""
```

Aus den Tabellen "Verkauf" und "Waren" werden die für die jeweilige Rechnung notwendigen Informationen zusammengestellt. In der Tabelle "Verkauf" ist die Anzahl der einzelnen Artikel aufgelistet, in der Tabelle "Waren" die Informationen über die "Ware" und den "Preis". Schließlich wird noch die "Anzahl" mit dem "Preis" multipliziert.

```
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
i = 0
WHILE oAbfrageergebnis.next
    loAnzahl = oAbfrageergebnis.getInt(1)
    stText = oAbfrageergebnis.getString(2)
    doPreis = oAbfrageergebnis.getDouble(3)
    doAnzahlPreis = oAbfrageergebnis.getDouble(4)
```

Die Daten sind in dem Objekt «oAbfrageergebnis» gespeichert. Sie werden Zeile für Zeile ausgelesen. Die Ziffern hinter **`getInt()`**, **`getString()`** usw. verweisen auf die jeweiligen Spalten der Abfrage.

```
    oTabelle.getCellByPosition(0,i).setValue(loAnzahl)
    oTabelle.getCellByPosition(1,i).setString(stText)
    oTabelle.getCellByPosition(2,i).setValue(doPreis)
    oTabelle.getCellByPosition(3,i).setValue(doAnzahlPreis)
```

Mit **`i`** wird die Zeile der Tabelle festgelegt. Die Zählung beginnt hier bei 0. Deswegen ist **`i`** vorher auch auf 0 festgesetzt worden. Der erste Zahlenwert bestimmt die Spalte in der Tabelle. Auf diese Art werden sämtliche am Anfang existierenden Zellen mit Werten (für Zahlen) oder mit Text gefüllt.

```
    oRows = oTabelle.getrows()
    oRows.insertByIndex(oRows.getCount(),1)
```

Die Anzahl der Zeilen der Tabelle wird ermittelt und anschließend eine neue Tabellenzeile hinzugefügt, die anschließend wieder befüllt werden kann.

```
    i = i + 1
WEND
```

Um den nächsten Datensatz in die neue Zeile zu schreiben muss zu **`i`** der Wert 1 addiert werden. Anschließend wird der nächste Datensatz ausgelesen.

```
oRows.removeByIndex(oRows.getCount()-1,1)
END SUB
```

Da das Makro in der Schleife immer schon eine zusätzliche Zeile anlegt muss beim letzten Schleifendurchgang eine Tabellenzeile zu viel angelegt worden sein. Diese Zeile wird anschließend wieder entfernt. Danach ist die Rechnung fertig erstellt.

Das Formular «Rechnung_Textfelder_Uebertrag»

Das Formular «Rechnung_Textfelder_Uebertrag» unterscheidet sich von dem Formular «Rechnung_Textfelder» äußerlich überhaupt nicht. Hinter dem Button Rechnung drucken wird lediglich ein etwas erweitertes Makro angesteuert.

Für die mehrzeiligen Warenangaben war außerdem eine kleine Änderung im Listenfeld des Formulars notwendig. Der zu dem Tabellenkopf «Ware» gehörige **Listeninhalt** wurde etwas erweitert:

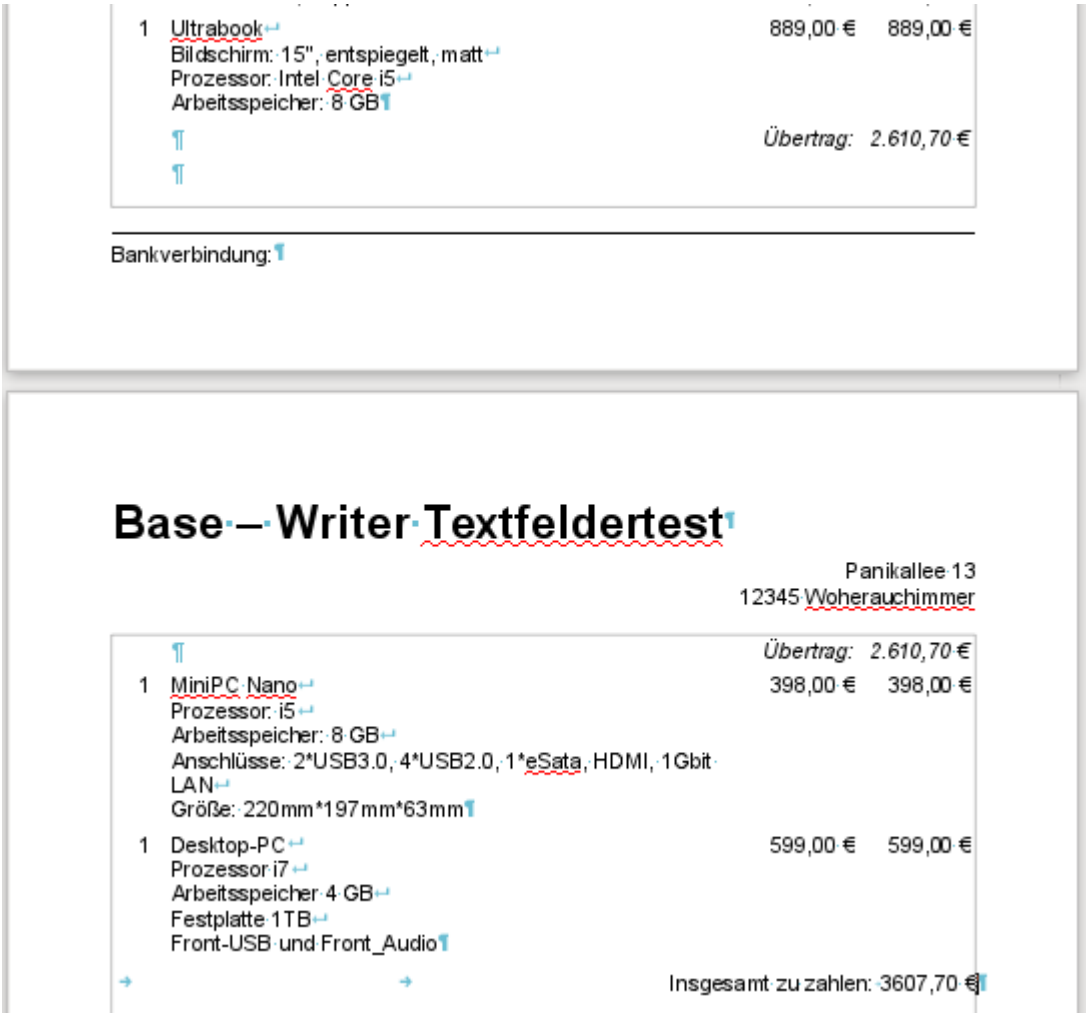
```
SELECT
   LEFT(REPLACE("Ware",CHAR(10),' ')|| SPACE(25), 25) || ' - ' ||
      REPLACE( RIGHT( SPACE(8) ||"Preis", 8),'.',',') || ' €',
      "ID"
   FROM "Waren"
   ORDER BY ("Ware" || ' - ' || "Preis" || ' €') ASC
```

In dem Feld "Ware" sind mehrzeilige Eingaben enthalten. Der Umbruch ist ein nicht sichtbares Zeichen. Das führt dazu, dass tatsächlich nur 24 und nicht 25 Zeichen angezeigt werden, wenn der

anzuzeigende Inhalt z.B. einen Umbruch hat. Das nicht sichtbare Zeichen, hier **`CHAR(10)`**, muss also entfernt werden. In dem obigen SQL-Code wird es durch ein Leerzeichen ersetzt.

Ausdruck aus dem Formular «Rechnung_Textfelder_Uebertrag»

Der Druck ergänzt die ursprüngliche Druckfunktion so, dass beim Übergang von einer zur nächsten Seite bei entsprechend vielen Rechnungstiteln ein Übertrag am Seitenschluss der vorhergehenden und am Seitenanfang der folgenden Seite erscheint.



Das Bild zeigt solch einen Übertrag beim Seitenwechsel. Der Übertrag berücksichtigt dabei auch, dass vielleicht mehrzeilige Eingaben bei den Waren erscheinen. Die in dem Screenshot enthaltenen Steuerzeichen zeigen, dass hier weiterhin die Tabelle genutzt wird und dem Übertrag am Seitenende wegen des zusätzlichen Platze noch eine leere Tabellenzeile folgt.

Makrosteuerung zum Ausdruck der Rechnung mit Übertrag

Als zusätzliche Information bei einer Rechnung über mehrere Seiten wird jetzt noch ermöglicht, einen Übertrag in die unterste Zeile der aktuellen Seite und in die oberste Zeile der nächstfolgenden Zeile zu schreiben.

Der Code des Makros «Rechnungsinhalt_zusammenstellen_Tabelle» wurde dazu entsprechend erweitert. Hier werden nur die Änderungen aufgezeigt.

```
SUB Rechnungsinhalt_zusammenstellen_Tabelle_Uebertrag
    ...
    DIM oCursor AS OBJECT
    DIM oTxtCursor AS OBJECT
```

```
DIM inStartSeite AS INTEGER
DIM inFolgezeilen AS INTEGER
DIM doUebertrag AS DOUBLE
...
doUebertrag = 0
inStartSeite = 1
inFolgezeilen = 1
oCursor = oNewDoc.CurrentController.ViewCursor
```

Die Variablen werden mit Anfangswerten belegt. Der Übertrag ist hat am Anfang den Wert 0, die erste Seite den Wert 1. In der Vorlage liegt eine Zeile unter der Tabelle, nämlich die Zeile mit dem Text «Insgesamt zu zahlen: ». Mit Hilfe des sichtbaren Cursors (**`ViewCursor`**) wird die Position innerhalb des Dokumentes bestimmt.

```
WHILE oAbfrageergebnis.next
    ...
    oCursor.JumpToLastPage()
    oCursor.JumpToEndOfPage()
    doUebertrag = doUebertrag + doAnzahlPreis
```

Der Cursor wird auf die letzte Seite und dort an das Ende bewegt. Der Übertrag wird laufend berechnet.

```
    IF oCursor.Page > inStartSeite THEN
        oCursor.goUp(inFolgezeilen,False)
```

Der Cursor wird vom Ende um die Zeilen zurückbewegt, die außerhalb der Tabelle noch mit im Dokument enthalten sind. Befindet sich der Cursor dann wieder auf der ersten Seite, so liegt er genau in einer leeren Tabellenzeile, die mit dem Übertrag gefüllt werden kann.

Zusätzlich wird auch noch ein Textcursor gebildet, mit dem der gerade erstelle Eintrag markiert wird und auf den Schriftschnitt «Italic» (kursiv) gesetzt wird. Dadurch ist der Übertrag besser von den anderen Einträgen unterscheidbar.

Anschließend wird noch eine Zeile eingefügt. Diese Zeile liegt jetzt auf der Folgeseite und kann dort mit dem Übertrag für den Seitenanfang gefüllt werden.

```
        IF oCursor.Page = inStartSeite THEN
            FOR ink = 1 TO 2
                i = i + 1
                oTabelle.getCellByPosition(2,i).setString("Übertrag:")
                oTxtCursor = oTabelle.getCellByPosition(2,i).createTextCursorByRange
                    (oTabelle.getCellByPosition(2,i).text)
                oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
                oTabelle.getCellByPosition(3,i).setValue(doUebertrag)
                oTxtCursor = oTabelle.getCellByPosition(3,i).createTextCursorByRange
                    (oTabelle.getCellByPosition(3,i).text)
                oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
                oRows = oTabelle.getRows()
                oRows.insertByIndex(oRows.getCount(),1)
            NEXT ink
        ELSE
```

Befindet sich die letzte neu eingefügte Tabellenzeile allerdings nicht auf der vorhergehenden Seite, weil eben mehrzeilige Tabelleneinträge im Bereich «Ware» den Umbruch stören, so sind die Zeilen für den Übertrag vor die entsprechende Tabellenzeile der folgenden Seite zu legen.

```
            i = i + 1
            oRows = oTabelle.getRows()
            oRows.insertByIndex(oRows.getCount()-2,1)
            oTabelle.getCellByPosition(2,i-1).setString("Übertrag:")
            oTxtCursor = oTabelle.getCellByPosition(2,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(2,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
            oTabelle.getCellByPosition(3,i-1).setValue(doUebertrag - doAnzahlPreis)
            oTxtCursor = oTabelle.getCellByPosition(3,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(3,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
            WHILE oCursor.Page = inStartSeite + 1
```

Nachdem die erste Tabellenzeile eingefügt wurde kann es sein, dass die folgende Zeile noch nicht auf der folgenden Seite erscheint. Dies hängt davon ab, wie viele Zeilen die Ware eingenommen hätte, für die eben auf der Vorseite nicht genug Platz war. In dieser Schleife sollen so viele Tabellenzeilen eingefügt werden, bis die nächste Zeile wirklich auf der Folgeseite liegt.

```
                oRows = oTabelle.getRows()
                oRows.insertByIndex(oRows.getCount()-2,1)
                oCursor.JumpToLastPage()
                oCursor.JumpToEndOfPage()
                oCursor.goUp(inFolgezeilen + 2,False)
                i = i + 1
            WEND
```

Nach dieser Schleife liegt die Tabellenzeile auf jeden Fall auf der Folgeseite. In Diese Tabellenzeile kann nun der Übertrag mit den entsprechenden Formatierungen eingetragen werden.

```
            i = i + 1
            oRows = oTabelle.getRows()
            oRows.insertByIndex(oRows.getCount()-2,1)
            oTabelle.getCellByPosition(2,i-1).setString("Übertrag:")
            oTxtCursor = oTabelle.getCellByPosition(2,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(2,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
            oTabelle.getCellByPosition(3,i-1).setValue(doUebertrag - doAnzahlPreis)
            oTxtCursor = oTabelle.getCellByPosition(3,i-1).createTextCursorByRange
                (oTabelle.getCellByPosition(3,i-1).text)
            oTxtCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
        END IF
```

Schließlich wird der Wert für die Startseite um 1 erhöht, damit die Prozedur erst bei einem erneuten Seitenumbruch den nächsten Übertrag generiert.

```
            inStartSeite = inStartSeite + 1
        END IF
        i = i + 1
    WEND
    oRows.removeByIndex(oRows.getCount()-1,1)
END SUB
```

Zum Schluss wird wieder die letzte Zeile der Tabelle, die leer geblieben ist, gelöscht.

Das Formular «Waren»

Das Formular «Waren» wurde nur benötigt, um für die Waren auch eine mehrzeilige Eingabe zu ermöglichen. Es bietet hier lediglich ein mehrzeiliges Textfeld, damit auch Absätze gespeichert werden. Für den Druck hat dieses Formular keine weitere Bedeutung.

Bilder in Base einbinden

Dass Bilder direkt in Base in eine Tabelle eingelesen werden können ist meist noch bekannt. Allerdings funktioniert die Verwaltung von Bildern in separaten Verzeichnissen genau so gut. Zusätzlich zur Bildaufnahme soll hier auch gezeigt werden, wie die Vorschau auf die Bilder an den Viewer des Betriebssystem übergeben werden kann, so dass nicht nur kleine Bildchen zu sehen sind und wie letztlich auch Bilder aus der Datenbank wieder nach außerhalb der Datenbank transportiert werden können.

Tabellen

Die Tabellen haben in dieser Datenbank die Funktion, entweder Bilder in den Tabellen zu speichern oder nur den Pfad zu Bildern aufzunehmen.

In der Tabelle "Bilder" werden folgende Felder definiert:

Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Beschreibung	Text	Soll einen Text aufnehmen, der als Bildbeschreibung dienen kann
Pfad	Text	Nimmt den Pfad zu dem Bild auf.

In der Tabelle "Bilder_intern" werden folgende Felder definiert:

Feldname	Feldtyp	Beschreibung
ID	Integer	Die ID ist Primärschlüssel dieser Tabelle.
Beschreibung	Text	Soll einen Text aufnehmen, der als Bildbeschreibung dienen kann
Bild	Bild	Nimmt das Bild als Binärdatenstrom auf.
BildName	Text	Sollte gegebenenfalls den Namen der Datei mit Dateiendung speichern.

Die Tabelle "Bilder_intern" steht in keiner Beziehung zur Tabelle "Bilder". Sie soll die interne Speichermöglichkeit von Bildern und den Umgang mit diesen Bildern aufzeigen.

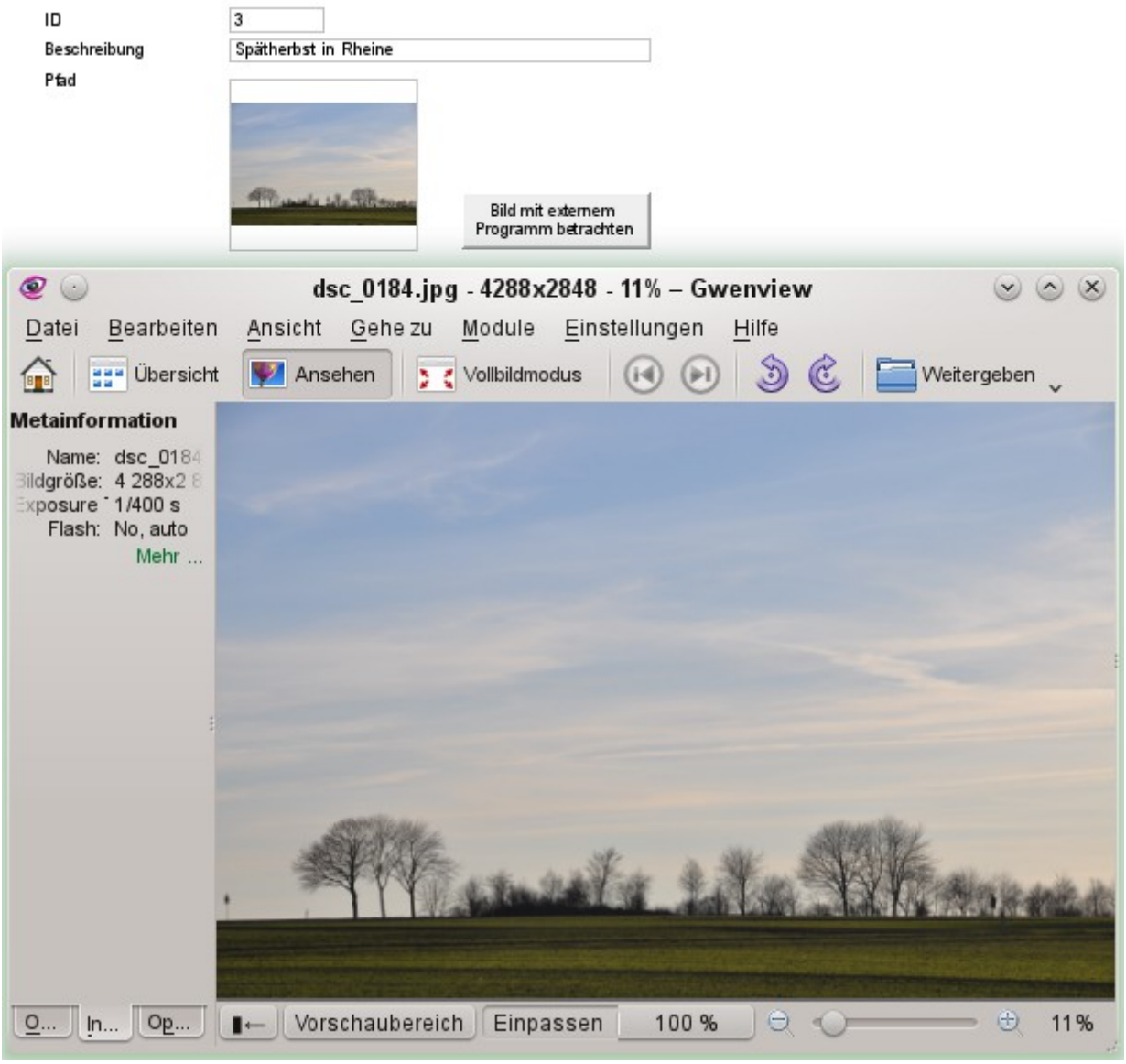
Formulare

Die Formulare sind notwendiger Bestandteil zur Aufnahme von Bildern in die Datenbank. Die Bildaufnahme geht nicht über die Tabellensicht oder Sicht einer Abfrage. Es wäre höchstens möglich, eine Aufnahme per Makro an den Formularen vorbei zu realisieren.

Auch zur Betrachtung der Bilder sind wieder Formulare erforderlich. Lediglich ein Bericht kann neben den Formularen noch Bilder in der Base-Datei darstellen.

Das Formular «Pfadeingabe»

Das Formular enthält neben den Eingabefeldern für «ID» und «Beschreibung» ein grafisches Steuerelement. Das grafische Steuerelement ist mit dem Feld "Pfad" aus der Tabelle verbunden. Wird durch Doppelklick auf das Feld ein Bild aus dem Dateisystem ausgesucht, so wird nur der Pfad (relativ zur Datenbankdatei) gespeichert. Das Bild wird trotzdem in dem grafischen Steuerelement angezeigt.



Das grafische Steuerelement bietet nicht die Möglichkeit, Details des Bildes zu betrachten. Deshalb wird hier über Bild mit externem Programm betrachten der im System angegebenen Dateibetrachter für die entsprechende Datei gestartet. Jetzt können Details betrachtet und alle weiteren Funktionen des Viewers genutzt werden.

Das Formular «Pfadeingabe_Tabellenkontrollfeld»

Wird statt der einzelnen Kontrollfelder ein Tabellenkontrollfeld genutzt, so kann das Bild nicht innerhalb des Tabellenkontrollfeldes dargestellt werden. Stattdessen wird hier der Pfad (relativ zur Datenbankdatei) direkt angezeigt. Das grafische Steuerelement ist hier direkt neben dem Tabellen-

kontrollfeld im selben Formular angeordnet. Es zeigt das Bild zu dem aktuellen markierten Datensatz an.

Makrosteuerung für die Formulare mit Pfadspeicherung

```
SUB Betrachten
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oShell AS OBJECT
    DIM stUrl AS STRING
    DIM stFeld AS STRING
    DIM arUrl_Start()
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    stUrl = oFeld.BoundField.getString
```

Das graphische Kontrollfeld im Formular wird aufgesucht. Da in der Tabelle nicht das Bild selbst, sondern nur der Pfad als Text gespeichert wird, wird hier über **`getString`** dieser Text ausgelesen.

Anschließend wird der Pfad zu der Datenbankdatei ermittelt. Mit **`oDoc.Parent`** wird die *.odb-Datei erreicht. Sie ist der Container für die Formulare. Über **`oDoc.Parent.Url`** wird schließlich die gesamte URL incl. Dateinamen ausgelesen. Der Dateiname ist auch zu sehen in **`oDoc.Parent.Title`**. Der Text wird jetzt mit der Funktion **`split`** aufgetrennt, wobei als Trenner der Dateiname benutzt wird. Die Auftrennung gibt so nur als erstes und einziges Element des Arrays den Pfad zur *.odb-Datei wieder.

```
    arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    stFeld = convertToUrl(arUrl_Start(0) + stUrl)
    oShell.execute(stFeld,,0)
END SUB
```

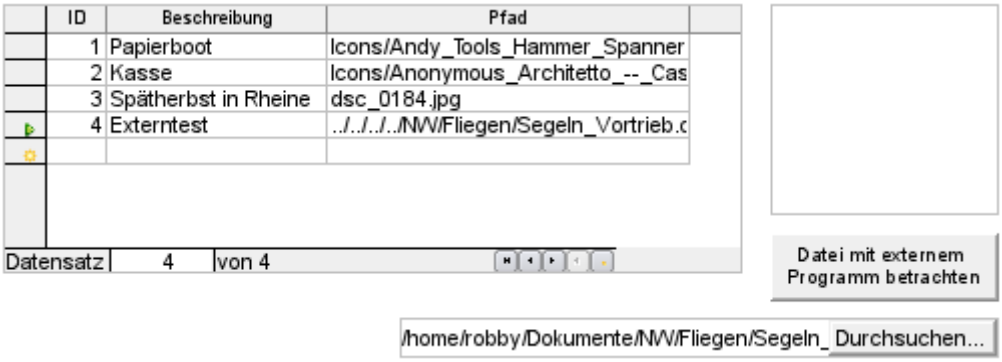
Externe Programme können über das Struct **`com.sun.star.system.SystemShellExecute`** gestartet werden. Dem externen Programm wird hier nur der Pfad zur Datei mitgegeben, der aus dem Pfad zur Datenbankdatei und dem intern gespeicherten relativen Pfad von der Datenbankdatei aus zusammengesetzt wurde. Die grafische Benutzeroberfläche des Betriebssystems entscheidet jetzt darüber, mit welchem Programm die entsprechende Datei zu öffnen ist.

Mit dem Kommando **`oShell.execute`** werden 3 Parameter übergeben. Als erstes wird eine ausführbare Datei oder der Pfad zu einer Datei aufgeführt, die im System mit einem Programm verbunden sind. Als zweites werden Parameter aufgeführt, mit denen das Programm gestartet werden soll. Als drittes wird über eine Ziffer mitgeteilt, wie mit Fehlermeldungen des Systems bei missglückter Ausführung umzugehen ist. Hier stehen 0 (Standard), 1 (keine Meldung anzeigen) und 2 (nur das Öffnen von absoluten URLs erlauben) zur Verfügung.

Das Formular «Pfadeingabe_Tabellenkontrollfeld_Pfadangabe»

Statt eines Bildes kann genauso gut der Pfad für eine Datei gespeichert werden. Wird anschließend der Button Datei mit externem Programm betrachten betätigt, so wird stattdessen das zu der Datei passende Programm des Betriebssystems ausgewählt.

Um den Pfad einer Datei aufzunehmen ist allerdings nicht das grafische Kontrollfeld geeignet. Hier wird mit dem Feld zur Dateiauswahl gearbeitet. Der Pfad zur ausgewählten Datei muss dann aber relativ zur Datenbank in dem entsprechenden Feld der Tabelle gespeichert werden.



Ergänzend zu dem vorhergehenden Formular ist hier ein Dateiauswahlfeld integriert. Dateiauswahlfelder zeigen die gerade erfolgte Auswahl an, sind aber nicht mit der Datenbank verbunden. Die Url des Dateiauswahlfeldes muss über ein Makro in einen relativen Link umgewandelt werden, damit die Datei passend über die Prozedur «Betrachten» geöffnet werden kann. Der Pfad in Zeile 4 zeigt an, dass von der Lage der Datenbankdatei aus 4 Verzeichnisse aufwärts gegangen werden muss und von dort aus das Verzeichnis «NW/Fliegen/» aufzusuchen ist.

Makrosteuerung für die relative Pfadangabe

Das folgende Makro ist an die Eigenschaft «Text modifiziert» des Dateiauswahlfeldes gebunden.

```
SUB PfadAuslesen
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM arUrl_Start()
    DIM ar()
    DIM ar1()
    DIM ar2()
    DIM stText AS STRING
    DIM stUrl_gesamt AS STRING
    DIM stUrl_Text AS STRING
    DIM stUrl AS STRING
    DIM stUrl_weg AS STRING
    DIM ink AS INTEGER
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    oFeld2 = oForm.getByName("Dateiauswahl")
```

Zuerst werden, wie bei allen Prozeduren, die Variablen deklariert. Anschließend werden die entsprechenden Felder aufgesucht, die für die Aufnahme des Pfades wichtig sind. Der gesamte anschließende Code wird nur dann ausgeführt, wenn in der Dateiauswahl auch Inhalt steht, das Feld also z.B. nach einem Datensatzwechsel nicht einfach geleert wird.

```
    IF oFeld2.Text <> "" THEN
        arUrl_Start = split(oDoc.Parent.Url,oDoc.Parent.Title)
        ar = split(ConvertToUrl(oFeld2.Text),"/")
        stText = ""
```

Der Pfad zur Datenbankdatei wird ausgelesen. Dies erfolgt, wie vorher schon erklärt, indem zuerst die gesamte Url gelesen wird und anschließend der Dateiname als Trenner eines Arrays dafür sorgt, dass im ersten teil des Arrays nur der direkte Pfad steht.

Anschließend werden alle Elemente des Pfades, der über die Dateiauswahl ausgesucht wurde, in das Array **`ar`** eingelesen. Als Trenner dient `/`. Das gelingt unter Linux direkt. Unter Windows muss der Inhalt von oFeld2.Text auf jeden Fall noch in eine URL umgewandelt werden, die mit dem Slash und nicht mit dem Backslash `\` arbeitet.

Die Trennung dient dazu, für die Ermittlung des Pfades zu der Datei einfach nur den Dateinamen hinten abzuschneiden. Daher wird im nächsten Schritt der Pfad zu der aufzurufenden Datei in der Variablen **stText** wieder zusammengesetzt. Die Schleife endet allerdings nicht mit dem höchsten Wert des vorher erstellten Arrays **ar**, sondern schon beim erreichen der vorhergehenden Wertes.

```
        FOR i = LBound(ar()) TO UBound(ar()) - 1
            stText = stText & ar(i) & "/"
        NEXT
        stText = Left(stText,Len(stText)-1)
        arUrl_Start(0) = Left(arUrl_Start(0),Len(arUrl_Start(0))-1)
```

Das zuletzt angehängte **/** wird wieder entfernt, da sonst im folgenden Array zum Schluss ein leerer Arraywert erzeugt wird, der den Pfadvergleich stören würde. Für einen richtigen Vergleich ist jetzt der Text noch zu einer tatsächlichen Url, beginnend mit **file:///**, umzuwandeln. Schließlich soll mit dem Pfad der Datenbankdatei verglichen werden, der genau auf dieser Basis zusammengesetzt ist.

```
        stUrl_Text = ConvertToUrl(stText)
        ar1 = split(stUrl_Text,"/")
        ar2 = split(arUrl_Start(0),"/")
        stUrl = ""
        ink = 0
        stUrl_weg = ""
```

In **ar1** werden alle Elemente gespeichert, die den Pfad zu der aufzurufenden Datei wieder geben. In **ar2** sind alle Elemente gespeichert, die den Pfad zur Datenbankdatei wieder geben. Anschließend wird das gesamte Array **ar2** schrittweise für einen Vergleich in einer Schleife durchlaufen.

```
        FOR i = LBound(ar2()) TO UBound(ar2())
            IF i <= UBound(ar1()) THEN
```

Nur wenn die Zahl **i** nicht größer ist als die Anzahl der in ar1 enthaltenen Elemente wird der folgende Code ausgeführt. Ist der Wert aus **ar2** gleich dem entsprechenden Wert aus **ar1** und ist bisher noch kein unterschiedlicher Wert aufgetaucht, dann wird der gemeinsame Inhalt in einer Variablen gespeichert, die zum Schluss von der Pfadangabe abgeschnitten werden kann.

```
                IF ar2(i) = ar1(i) AND ink = 0 THEN
                    stUrl_weg = stUrl_weg & ar1(i) & "/"
                ELSE
```

Ist irgendwann ein Unterschied zwischen den beiden Arrays aufgetaucht, so wird für jeden unterschiedlichen Wert die Bezeichnung für «setze ein Verzeichnis höher an» in der Variablen **stUrl** gespeichert.

```
                    stUrl = stUrl & "../"
                    ink = 1
                END IF
```

Sobald der Index, der durch die Variable **i** dargestellt wird, größer wird als die Anzahl der Elemente von **ar1**, wird für jeden weiteren Wert von **ar2** wieder ein weiteres **../** in der Variablen **stUrl** gespeichert.

```
            ELSE
                stUrl = stUrl & "../"
            END IF
        NEXT
        stUrl_gesamt = ConvertToUrl(oFeld2.Text)
        oFeld.boundField.UpdateString(stUrl & Right(stUrl_gesamt,Len(stUrl_gesamt)-
            Len(stUrl_weg)))
    END IF
END SUB
```

Ist die Schleife durch **ar2** komplett durchlaufen, so steht fest, ob und wie viele Verzeichnisse höher von der Datenbankdatei aus gesehen das Verzeichnis für die aufzurufende Datei liegt. Jetzt wird einmal die **stUrl_gesamt** aus dem Text im Dateiauswahlfeld erstellt. Diese enthält auch den Dateinamen. Anschließend wird in das graphische Feld der Wert für die Url übertragen. Der Wert für die Url beginnt mit **stUrl**, in der **../** in der erforderlichen Anzahl stehen. Dann wird von

stUrl_gesamt der vordere Teil abgeschnitten, der bei den Pfaden zur Datenbankdatei und zur externen Datei gleich war. Der weg zu schneidende Inhalt wurde in **stUrl_weg** gespeichert.

Die Formulare «Bildaufnahme» und «Bildaufnahme_Name»

Äußerlich unterscheiden sich diese Formulare nicht vom Formular «Pfadeingabe». Das Formular «Bildaufnahme_Name» hat lediglich ein zusätzliches Feld, in dem der Name der Bilddatei abgespeichert werden kann, also z.B. «Haus.jpg». Dies könnte für Betriebssysteme, die nach der Dateiendung entsprechende Programme für das Öffnen von Dateien bereitstellen, wichtig sein, da der eigentliche Bildname nach dem Speichern der Bilddaten in der Datenbank nicht mehr nachvollzogen werden kann.

An dieser Stelle sollte aber gleichzeitig darauf hingewiesen werden, dass das Abspeichern von Bilddaten sehr schnell eine Datenbank von der Größe her deutlich aufbläht. Deshalb sollten intern möglichst nur kleinere Bilder, z.B. Icons, aber nicht komplette Fotos aktueller Digitalkameras abgespeichert werde.

Makrosteuerung für das Formular «Bildaufnahme»

Der wichtigste Unterschied zu der externen Bildverwaltung ist gleichzeitig auch die entscheidende Frage für den Datenbanknutzer: Wie komme ich eigentlich an die Bilddaten wieder heran, die jetzt in der Tabelle der Datenbank gespeichert liegen. Mit einem externen Viewer kann ich schließlich nicht interne Bilder ansehen. Die internen Bilder müssen dafür zumindest vorübergehend ausgelesen werden um damit über ein Betrachtungsprogramm drauf zugreifen zu können.

```
SUB BildAuslesen
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oStream AS OBJECT
    DIM oShell AS OBJECT
    DIM oPath AS OBJECT
    DIM oSimpleFileAccess AS OBJECT
    DIM st AS STRING
    DIM stPfad AS STRING
    DIM stFeld AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld = oForm.getByName("GraphischesFeld")
    oStream = oFeld.BoundField.getBinaryStream
```

Mit **getBinaryStream** wird aus dem mit dem graphischen Kontrollfeld verbundenen Feld der Datenbanktabelle der Binärcode ausgelesen, der zu dem Bild gehört.

```
    oPath = createUnoService("com.sun.star.util.PathSettings")
    st = ""
```

Es kann, je nach Betriebssystem, notwendig sein, eine Dateiendung dem jeweils gewählten Namen beizufügen. In der Variablen «st» kann so eine Endung aufgeführt werden (z. B. ".png" oder ".jpg"). Aus den Pfadangaben in **Extras → Optionen → LibreOffice → Pfade** wird der Pfad für temporäre Dateien ausgelesen. Dorthin soll das Bild gespeichert werden.

```
    stPfad = oPath.Temp & "/DbBild" & st
    oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    oSimpleFileAccess.writeFile(stPfad, oStream)
```

Mit dem Struct **com.sun.star.ucb.SimpleFileAccess** wird die ausgelesene Datei in das temporäre Verzeichnis geschrieben. Der weitere Verlauf des Makros ist wieder identisch mit der Prozedur «Betrachten» für die externen Bilder.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    stFeld = convertToUrl(stPfad)
    oShell.execute(stFeld,,0)
```

```
END SUB
```

Makrosteuerung für das Formular «Bildaufnahme_Name»

In diesem Makro wird lediglich statt eines fest vorgegebenen Dateinamens die Möglichkeit geboten, einen Dateinamen mit Endung aus der Datenbank auszulesen. Hier nur die Unterschiede zur Prozedur «BildAuslesen»

```
SUB BildAuslesen_mitName
    ...
    oFeld2 = oForm.getByName("BildName")
    stName = oFeld2.Text
    IF stName = "" THEN
        stName = "DbBild"
    END IF
```

Steht in dem Formularfeld "BildName" eine Bezeichnung für das zu speichernde Bild, so wird das Bild unter dem entsprechenden Namen abgespeichert. Damit besteht die Möglichkeit, der erzeugten Datei eine Endung mitzugeben und außerdem auch noch direkt nacheinander mehrere Bilder abzuspeichern, ohne die vorhergehenden zu überschreiben. Die gespeicherten Bilder können anschließend aus dem temporären Verzeichnis ist das entsprechende Wunschverzeichnis übertragen werden.

```
    ...
    stPfad = oPath.Temp & "/" & stName
    ...
END SUB
```

Berichte

Übersichtliche Ausdrucke der Daten zusammen mit Bildern sind ohne Probleme über den Report-Builder möglich.

ID	1	
Beschreibung	Papierboot	
Pfad	Icons/Andy_Tools_Hammer_Spanner.png	
ID	2	
Beschreibung	Kasse	
Pfad	Icons/Anonymous_Architetto_--_Cassa_d_epoca.png	
ID	3	
Beschreibung	Spätherbst in Rheine	
Pfad	dsc_0184.jpg	

Die Bilder werden, wie im Formular, in einem grafischen Kontrollfeld dargestellt. Das Kontrollfeld ist so voreingestellt, dass das Seitenverhältnis beibehalten wird. Sonst könnte es dazu kommen, dass Bilder nicht mehr zu erkennen sind, weil nur ein Ausschnitt von ihnen gezeigt oder das Bild verzerrt in den vorgesehenen Rahmen eingepasst wird.

Mit den Standardeinstellungen kann es dazu kommen, dass eine Datei, die nicht im grafischen Kontrollfeld angezeigt werden kann, dafür sorgt, dass kein einziges Bild erscheint. Hier kann auf zweierlei Weise gegengesteuert werden:

Der Bericht wird zum Editieren geöffnet. Das grafische Steuerelement wird markiert. In **Eigenschaften → Allgemein → Als Verknüpfung vorbereiten** wird «Nein» ausgewählt. Alle Bilder erscheinen dann problemlos. Bei anderen Dateien wie z.B. einer *.pdf-Datei erscheint dann ein kleines Rechteck mit einer Meldung, dass diese Datei nicht geladen werden konnte.

Für eine kombinierte Speicherung von Bildern und Dateien kann stattdessen auch dafür gesorgt werden, dass die nicht anzeigbaren Dateien gar nicht erst an den Report-Builder weiter gegeben werden. Der Bericht könnte z.B. auf der folgenden Abfrage aufbauen:

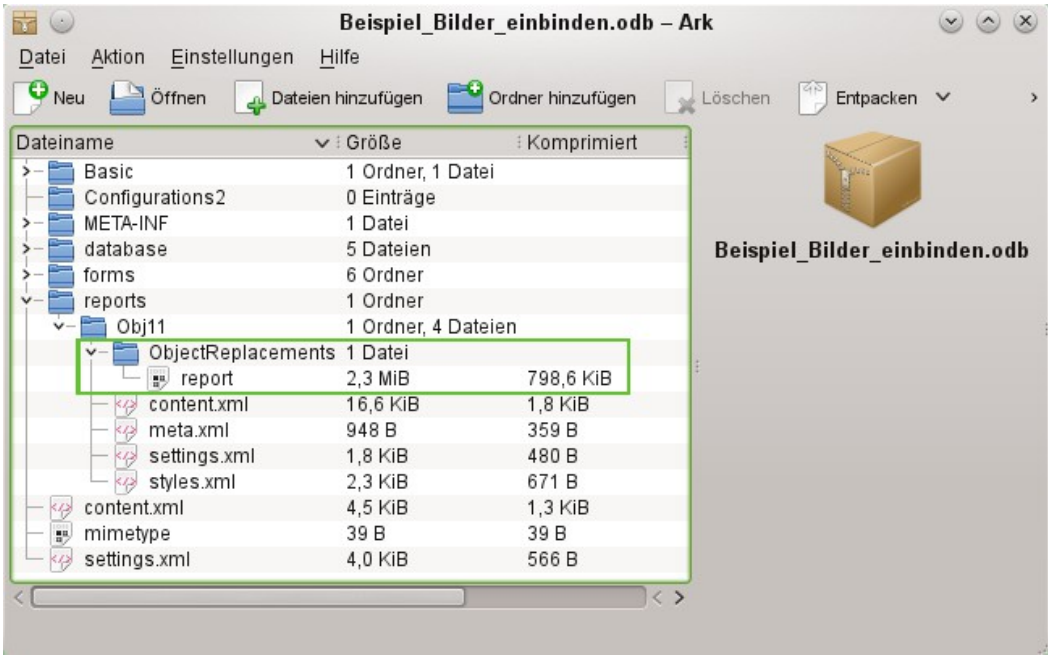
```
SELECT * FROM "Bilder"
   WHERE UPPER ( "Pfad" ) LIKE '%.JPG'
      OR UPPER ( "Pfad" ) LIKE '%.PNG'
```

Mit diesem Code würden nur *.jpg- und *.png-Bilder angezeigt. Damit wäre vermutlich schon ein Großteil der üblichen Formate erfasst.

Jetzt ist der Bericht wieder zum Bearbeiten zu öffnen. Über den Report-Navigator wird durch einen Klick auf den obersten Eintrag «Bericht» in den **Eigenschaften → Daten → Art des Inhaltes** auf «Abfrage» umgestellt und in **Daten → Inhalt** die entsprechende Abfrage ausgesucht.

Hinweis

Nach dem erneuten Editieren von Berichten mit Bildern fällt auf, dass die Datenbankdatei deutlich größer wird. Bei den Tests zu dieser Beschreibung mit großen Bilddateien entstand plötzlich eine Datei der Größe 16 MB. Innerhalb der *.odb-Datei legt Base aus nicht nachvollziehbaren Gründen im Berichtsverzeichnis einen Ordner «ObjectReplacements» an. Dieser Ordner enthält dann eine Datei «report», die für eine entsprechende Vergrößerung der *.odb-Datei sorgt.



Wenn die Datenbankdatei in einem Packprogramm geöffnet wird ist dieser Ordner mit Inhalt im Verzeichnis «reports» im Unterverzeichnis zu dem jeweiligen Bericht sichtbar. Dieses Verzeichnis kann über das Packprogramm gefahrlos gelöscht werden.

Wenn Berichte nicht wiederholt editiert werden, reicht ein einmaliges Löschen des Verzeichnisses aus.

Der Bug ist hier gemeldet: *https://bugs.freedesktop.org/show_bug.cgi?id=80320*

Mailversand aus einer Datenbank heraus

In Foren kam immer wieder der Wunsch auf, von einer Datenbank aus die in einer Adressverwaltung gespeicherten Mailadressen auch direkt zum Mailversand nutzen zu können. Zum Mailversand wird in dieser Datenbank immer das Mailprogramm genutzt, das über das Betriebssystem zur Verfügung gestellt wird. In der Datenbank kann aber der gesamte Mailverkehr abgelegt werden. Nur die letzte Kontrolle, ob denn nun die E-Mail tatsächlich vom Mailprogramm abgeschickt wurde, ist von Base heraus nicht möglich.

Tabellen

Die Datenbank besteht aus drei Tabellen.

In der Tabelle "Kontakte" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt wer-

		den.
Name	Text	Der Name der Person, an die die Mail verschickt werden soll und zu der auch die Webadresse gehört.
E_Mail	Text	Die E-Mail-Adresse der Person
Web	Text	Falls die Person eine Website führt, so kann z.B. auch aus Base heraus die Website aufgerufen werden.

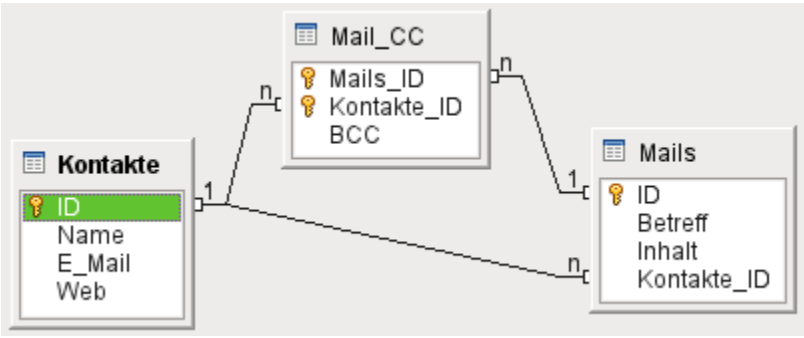
In der Tabelle "Mails" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld sollte als Auto-Wert-Feld gesetzt werden.
Betreff	Text	Betreff der zu versendenden E-Mail
Inhalt	Text	Der Inhalt der Mail kann viel Platz einnehmen. Die Standardeinstellung des Feldes mit 100 Zeichen dürfte also in den wenigsten Fällen ausreichen. In dem Beispiel ist dies auf 1000 Zeichen erweitert worden
Kontakte_ID	Integer	Falls die Person eine Website führt, so kann z.B. auch aus Base heraus die Website aufgerufen werden.

In der Tabelle "Mail_CC" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
Mails_ID	Integer	Die Mails_ID wird als Fremdschlüssel aus der Tabelle "Mails" bezogen. Dadurch können mehrere Personen eine Mailkopie erhalten.
Kontakte_ID	Integer	Die Kontakte_ID dienst dazu, die Mailadresse von anderen Personen aus der Tabelle "Kontakte" zu ermitteln. Mails_ID und Kontakte_ID bilden zusammen den Primärschlüssel der Tabelle "Mail_CC"
BCC	Ja/Nein	Muss nur angeklickt werden, wenn die Person eine Blindkopie erhalten soll.

Alle drei Tabellen werden in den Beziehungen noch miteinander verbunden.



Wechselt eine Person im Laufe der Zeit die E-Mail-Adresse, so hat das keinen weiteren Einfluss auf die gespeicherten E-Mails. Nur der Inhalt sowie die Verbindung zu entsprechenden personen wird gespeichert. Einmal überschriebene Mailadressen erscheinen nicht in der Datenbank, sondern nur noch in der Liste der versandten Mails des Mailprogramms.

Einem Kontakt können beliebig viele Mails zugeordnet werden. Das Verhältnis von "Kontakte" zu "Mails" ist ein Verhältnis 1:n.

Einer Mail können beliebig viele Kopieempfänger zugeordnet werden. Das Verhältnis von "Mails" zu "Mail_CC" ist 1:n.

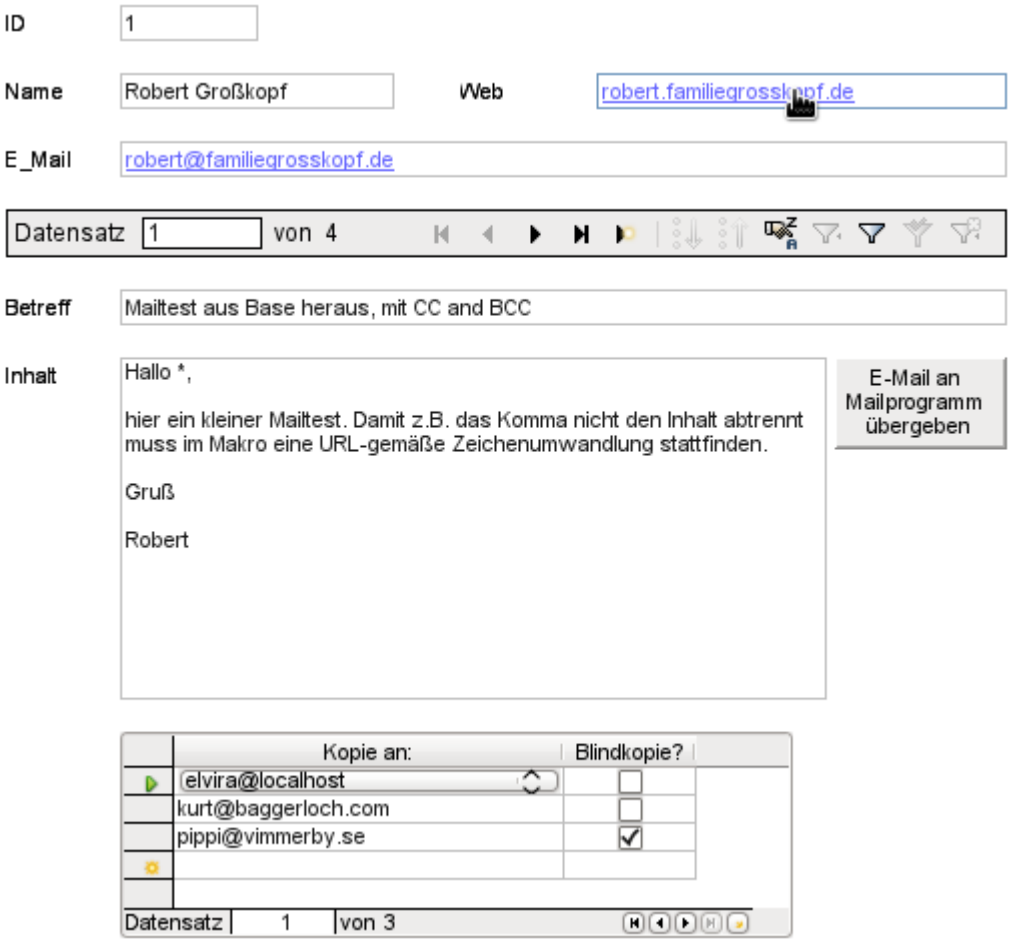
Ein Kontakt kann auch mehrmals als Kopieempfänger zugeordnet werden. Das Verhältnis von "Kontakte" zu "Mail_CC" ist 1:n.

Ausgeschlossen ist aber, dass eine Person mehrere Kopien der gleichen Mail erhält. "Mails_ID" und "Kontakte_ID" sind zusammen Primärschlüssel und müssen einzigartig sein.

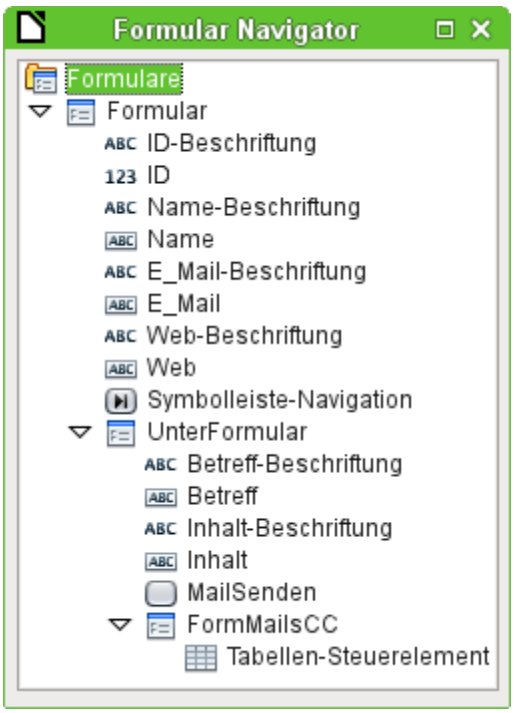
Formular

Das Beispiel ist in einem einzigen Formular zusammen gefasst. In dem Formular werden alle drei Tabellen benötigt, damit eine E-Mail mit allen zur Verfügung stehenden Funktionen an das Mailprogramm weiter gegeben werden kann.

Das folgende Bild zeigt dabei zwei Felder, die bereits durch die Formatierung der Schrift (unterstrichen und blau) als Felder mit einem Link gekennzeichnet sind. Felder, die mit einem Link direkt funktionieren, gibt es in Base nicht. Um trotzdem eine entsprechende Funktion bereit zu stellen ist ein Eingriff über Makros erforderlich.



Das Formular ist von oben nach unten in ein Hauptformular, ein Unterformular und ein Unterformular des Unterformulars gegliedert. Die entsprechenden Gliederung ist aus der Ansicht des Formularnavigators zu ersehen:



Sämtliche Einträge oberhalb der Navigationsleiste werden über «Formular» in der Tabelle «Kontakte» gespeichert. Das Formular «Formular» ist mit «UnterFormular» über das Feld "Kontakte"."ID" mit dem Feld "Mails"."Kontakte_ID" verknüpft. In «UnterFormular» werden die gesamten Mails abgespeichert und auch der Mailversand gestartet. Lediglich die Kopien der Mails müssen in einem Unterformular des Unterformulars, dem Formular «FormMailsCC», verwaltet werden. Schließlich soll es möglich sein, beliebig viele Mailadressen auszuwählen, an die eine Kopie der Mail gehen soll. «UnterFormular» und «FormMailsCC» sind über die Felder "Mails"."ID" und "Mail_CC"."Mails_ID" miteinander verknüpft.

Makrosteuerung für den Mauszeiger über einem Link

Der Mauszeiger wird über den entsprechenden **`UnoService`** beeinflusst. Die Typen für den Mauszeiger sind in **`com.sun.star.awt.SystemPointer`** verzeichnet. Die 27 verweist hier auf das Aussehen des Zeigers als Hand. Dieses Makro wird dann ausgelöst, wenn sich die Maus innerhalb des Formularfeldes befindet: **Textfeld → Eigenschaften → Ereignisse → Maus innerhalb**. Da das Ereignis direkt von dem Formularfeld heraus gesteuert wird (und nicht durch einen Button von außerhalb) muss hier nicht erst das Formularfeld aufgesucht werden. Das Formularfeld ist Urheber des Ereignisses, **`oEvent.source`**.

```
SUB Mauszeiger(oEvent AS OBJECT)
    DIM oPointer AS OBJECT
    oPointer = createUnoService("com.sun.star.awt.Pointer")
    oPointer.setType(27)
    oEvent.Source.Peer.SetPointer(oPointer)
END SUB
```

Makrosteuerung für den Programmstart mit Link

Mit dieser Prozedur wird eine Website im Browser oder eine Mailadresse als Empfängeradresse im E-Mail-Programm der grafischen Benutzeroberfläche des Betriebssystems geöffnet. Die Art, eine Datei über das damit verknüpfte Programm zu öffnen, lässt sich auf alle möglichen Dateitypen übertragen, für die eben ein Programm zur Verfügung steht.

```
SUB Website_Aufruf(oEvent AS OBJECT)
    DIM oFeld AS OBJECT
    DIM oShell AS OBJECT
    DIM stFeld AS STRING
```

```
    oFeld = oEvent.Source.Model
```

Ausgangspunkt ist wie beim Mauszeiger direkt das Feld, in dem der Klick mit der Maus erfolgt. Das Makro wird über **Textfeld → Eigenschaften → Ereignisse → Maustaste gedrückt** ausgelöst. Das Feld ist über diesen Weg bereits bekannt. Der Text, den das Feld anzeigt, kann ausgelesen werden. Enthält das Feld keinen Text, so braucht auch kein Browser oder Mailprogramm gestartet zu werden. Die Prozedur wird also bei einem leeren Feld abgebrochen.

```
    stFeld = oFeld.Text
    IF stFeld = "" THEN
        EXIT SUB
    END IF
```

Enthält der Text ein «@», so handelt es sich um eine E-Mail-Adresse. Die Gültigkeit der Adresse wird hier nicht weiter überprüft. Für eine E-Mail-Adresse startet der Aufruf des Mailprogramms mit **`mailto:`**.

```
    IF InStr(stFeld,"@") THEN
        stFeld = "mailto:"+stFeld
```

Beginnt der Text mit **`http://`**, so handelt es sich um eine vollständige Internetadresse. Solch eine Adresse kann direkt mit **`convertToUrl`** in eine Schreibweise umgewandelt werden, die auch Url-konform ist. Das sonst übliche **`file:///`** wird durch die Funktion nicht vor die Adresse gesetzt.

```
    ELSEIF InStr(stFeld,"http://") THEN
        stFeld = convertToUrl(stFeld)
    ELSE
        stFeld = "http://"+stFeld
        stFeld = convertToUrl(stFeld)
    END IF
```

Ist in dem Text weder @ noch **`http://`** enthalten, so wird hier davon ausgegangen, dass es sich um eine Internetadresse ohne den entsprechenden Vorspann handelt. Entsprechend wird die Adresse mit einem **`http://`** ergänzt. Würde dies nicht erfolgen, so würde die Adresse wie ein Dateipfad auf dem eigenen Rechner interpretiert. Bei einem tatsächlich existierenden Pfad zu z. B. einer *.png-Datei würde dann ein Bildbetrachtungsprogramm geöffnet.

Auch für die Internetadressen gilt, dass sie innerhalb des Makros nicht auf Gültigkeit überprüft werden. Es wird zum Schluss lediglich die entsprechende Adresse an die Benutzeroberfläche weiter gegeben, die dann ein dafür entsprechendes Programm sucht und öffnet.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    oShell.execute(stFeld,,0)
END SUB
```

Makrosteuerung für den Aufruf des Mailprogramms mit Parametern

Über den Druck auf den Button E-Mail an Mailprogramm übergeben werden Adresse, Betreff, Inhalt, Kopieadressen und Blindkopieadressen an das Mailprogramm weiter gereicht. Der Mailaufruf mit **`mailto:Empfänger?subject=...&body=...&cc=...&bcc=...`** . Anhänge sind laut Definition von **`mailto`** nicht definiert. Manchmal funktioniert allerdings trotzdem **`attachment=...`** . Bei dem Mailprogramm Thunderbird ist **`attachment=...`** aus Sicherheitsgründen vom Betrieb mit **`mailto`** ausgenommen. Am Beispiel von Thunderbird wird noch gezeigt, wie auch der Versand mit Mailanhängen gelingen kann. Das Formular müsste dazu aber sinnvollerweise noch mindestens mit einem Dateiauswahlfeld ergänzt werden, da vermutlich nicht jede Person den gleichlautenden Mailanhang erhalten soll.

```
SUB Mail_Aufruf
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM oFeld4 AS OBJECT
    DIM oShell AS OBJECT
```

```
DIM oColumns AS OBJECT
DIM oDatenquelle AS OBJECT
DIM oVerbindung AS OBJECT
DIM oSQL_Anweisung AS OBJECT
DIM oAbfrageergebnis AS OBJECT
DIM inIndex AS INTEGER
DIM i AS INTEGER
DIM loMails_ID AS LONG
DIM stFeld1 AS STRING
DIM stFeld2 AS STRING
DIM stFeld3 AS STRING
DIM stFeld4 AS STRING
DIM stSql AS STRING
DIM stMail_CC AS STRING
DIM stMail_BCC AS STRING
DIM st_CC AS STRING
DIM st_BCC AS STRING
DIM arSo_Zeichen()
DIM arSo_Ersatz()
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
```

Nachdem alle Variablen deklariert sind wird der Pfad zu den verschiedenen Feldern geklärt. Anschließend wird in dem «UnterFormular», dem die Tabelle "Mails" zugrunde liegt, das Feld "ID" aufgesucht. Ein über **`oSubForm.Columns`** angesprochenes Feld muss dafür nicht in dem Formular als Feld hinterlegt sein. Es ist in der Datenquelle des Formulars enthalten und kann deshalb für den aktuellen Datensatz ausgelesen werden.

```
oForm = oDrawpage.Forms.getByName("Formular")
oFeld1 = oForm.getByName("E_Mail")
oSubForm = oForm.getByName("UnterFormular")
oFeld2 = oSubForm.getByName("Betreff")
oFeld3 = oSubForm.getByName("Inhalt")
stFeld1 = oFeld1.Text
oColumns = oSubForm.Columns
inIndex = oSubForm.findColumn("ID")
loMails_ID = oSubForm.getLong(inIndex)
IF stFeld1 = "" THEN
   msgbox "Keine Mailadresse vorhanden." & CHR(13) &
      "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
   EXIT SUB
END IF
```

Enthält der Datensatz aus der Tabelle "Kontakte" keine E-Mail-Adresse, so wird die Prozedur direkt abgebrochen. Anschließend wird im «Betreff» und im «Inhalt» der Text in einen Url-konformen Text umgewandelt, da sonst Sonderzeichen und Zeilenumbrüche nicht übernommen werden. Der Eintrag **`file:///`** wird dabei automatisch hinzugefügt und muss wieder entfernt werden. Der eigentliche Text beginnt also ab dem 9. Zeichen des Ulr-konformen Textes.

```
stFeld3 = oFeld3.Text
stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

Leider greift die Konvertierung zur Url bei einigen Sonderzeichen nicht. Nach Kommas bricht so z.B. der Text ab. Deswegen muss der bereits über die Funktion **`ConverToUrl`** bearbeitete Text noch einmal auf verschiedene Zeichen durchsucht werden. Über zwei Arrays werden hier noch diverse Sonderzeichen zusätzlich getestet und ersetzt. Das erste Array enthält solche Sonderzeichen, das zweite Array an genau der gleichen Arrayposition die Url-konforme Schreibweise im Hexadezimalcode. Der Text für den Mailinhalt wird hier an jeder Stelle, an der ein Zeichen aus **`arSo_Zeichen`** vorkommt, in Einzelteile zerschnitten. Anschließend wird das Ergebnis wieder mit dem entsprechenden Zeichen aus **`arSo_Ersatz`** zusammengefügt.

```
arSo_Zeichen = Array("\","^","~","[","]"," ","&","+",",",";",":","=","?","@")
arSo_Ersatz = Array("5C","5E","7E","5B","5D","20","26","2B","2C","3B",
   "3A","3D","3F","40")
FOR i = LBound(arSo_Zeichen()) TO UBound(arSo_Zeichen())
   stFeld3 = Join(Split(stFeld3,arSo_Zeichen(i)),"%" & arSo_Ersatz(i))
```

```
NEXT
```

Die Variablen für das CC(Kopie) bzw. BCC (Blindkopie) werden als leere Textvariablen vordefiniert. Die Variablen **st_CC** und **st_BCC** sind hier nur eingefügt worden, die Behandlung dieser Felder direkt über den Shellaufruf von Thunderbird anders funktioniert als mittels **mailto**.

```
stMail_CC = ""
stMail_BCC = ""
st_CC = ""
st_BCC = ""
oDatenquelle = ThisComponent.Parent.CurrentController
IF NOT (oDatenquelle.isConnected()) THEN
   oDatenquelle.connect()
END IF
oVerbindung = oDatenquelle.ActiveConnection()
oSQL_Anweisung = oVerbindung.createStatement()
stSql = "SELECT ""Kontakte"".""E_Mail"", ""Mail_CC"".""BCC"" FROM ""Mail_CC"",
   ""Kontakte"""
stSql = stSql + " WHERE ""Mail_CC"".""Kontakte_ID"" = ""Kontakte"".""ID"" AND
   ""Mail_CC"".""Mails_ID"" = '"+ loMails_ID +"'"
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
WHILE oAbfrageergebnis.next
   IF oAbfrageergebnis.getBoolean(2) = TRUE THEN
      stMail_BCC = stMail_BCC + oAbfrageergebnis.getString(1) + ","
   ELSE
      stMail_CC = stMail_CC + oAbfrageergebnis.getString(1) + ","
   END IF
WEND
```

Über eine Abfrage werden die Mailadressen ermittelt, an die ein Kopie bzw. Blindkopie weitergereicht werden soll. Eine Abfrage ist hier notwendig, da über das Formular nur auf genau einen Datensatz zugegriffen werden kann. Anschließend werden die ausgelesenen Werte für das abschließende Kommando zusammengestellt, mit dem das Mailprogramm gestartet werden soll. Hier ist eine alternative Formulierung für Thunderbird direkt mit dem Shell-Kommando aufgeführt.

```
IF stMail_BCC <> "" THEN
   st_BCC = Left(stMail_BCC,Len(stMail_BCC)-1)
   stMail_BCC = "&bcc="+st_BCC
   st_BCC = ",bcc='"+st_BCC+"'"
END IF
IF stMail_CC <> "" THEN
   st_CC = Left(stMail_CC,Len(stMail_CC)-1)
   stMail_CC = "&cc="+st_CC
   st_CC = ",cc='"+st_CC+"'"
END IF
oShell = createUnoService("com.sun.star.system.SystemShellExecute")
oShell.execute("mailto:"+stFeld1+"?subject="+stFeld2+"&body="+stFeld3
   +stMail_CC+stMail_BCC,,0)
```

Die Variablen **stMail_CC** sowie **stMail_BCC** sind nicht immer mit Inhalt belegt. Durch die in der vorhergehenden Zeile erfolgte Formulierung wird vermieden, dass ein «&cc=» ohne irgendeinen Inhalt in der Ausführung erscheint.

Für Thunderbird wird hier noch gezeigt, wie der Aufruf mit Parametern erfolgen kann. Dieser Aufruf ermöglicht es schließlich auch zusätzlich noch Anhänge an das Mailprogramm weiter zu geben. Der Aufruf von Thunderbird in den verschiedenen Betriebssystemen ist unterschiedlich. Hier wird nur der funktionierende Aufruf aus einer Linux-Umgebung gezeigt. Details zu dem Aufruf mit Parametern siehe auch: *http://www.thunderbird-mail.de/wiki/Aufrufparameter_von_Thunderbird*

```
Shell("thunderbird -compose ""to="+stFeld1+st_CC+st_BCC+",subject="+stFeld2+
   ",body="+stFeld3+",attachment='file:///home/user/Testdatei.pdf'""")
```

Statt des **UnoServices** wird hier direkt mit dem Kommando **Shell()** die Kommandozeile des Systems angesprochen. Die angehängte Datei ist hier direkt im Code enthalten und müsste bei Nutzung dieser Funktion natürlich über das Formular ermittelt werden. Für nur einen Anhang reicht ein Feld in der Tabelle "Mails", für mehrere Anhänge müsste aber ähnlich wie bei der Tabelle

"Mail_CC" eine Tabelle "Anhang" erstellt werden. Erst so lassen sich mehrere Anhänge mit einer Mail verschicken.

```
END SUB
```

Der Shell-Befehl ist in dem der Beispieldatenbank beigefügten Makro natürlich als Kommentar geschrieben, so dass das Makro mit jedem in dem Betriebssystem verankerten Mailprogramm ohne Fehlermeldungen zu Ende laufen müsste.

Suchen und Filtern von Daten ohne Makroeinsatz

Base selbst bietet sowohl eine Suchfunktion als auch eine Möglichkeit, Daten zu filtern.

Beim Suchen wird schrittweise der gesamte Datenbestand einer Tabelle durchlaufen und die Treffer nacheinander markiert. Die Anzahl der Ergebnisdatensätze wird nicht auf Ergebnisse eingeschränkt, auf die das Suchkriterium zutrifft.

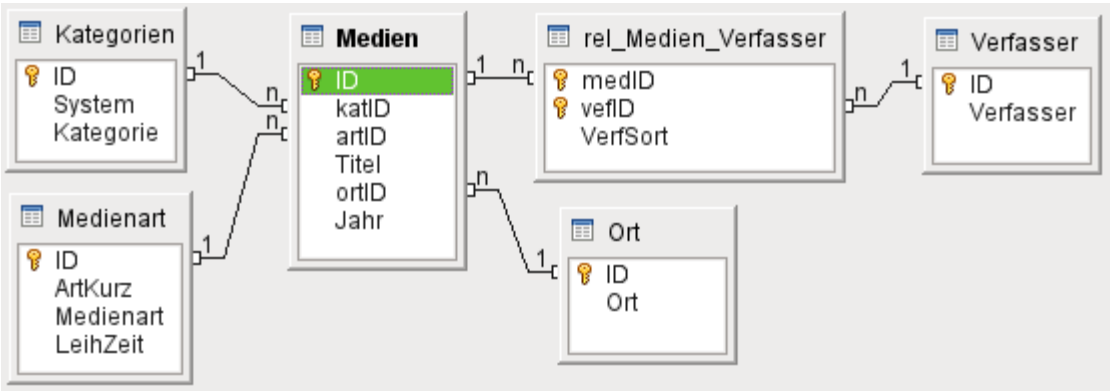
Beim Filtern wird durch einen oder mehrere Filterbegriffe direkt die Ergebnismenge aus der Tabelle heraus beeinflusst. Taucht das gewünschte Kriterium nur bei 3 von 100 Datensätzen auf, so werden eben nur 3 Datensätze angezeigt. Die Suchfunktion hingegen würde die 100 Datensätze durchlaufen und die einzelnen Treffer nacheinander markieren.

Die eingebaute Suchfunktion ist allerdings recht langsam, die Datenfilterung erfordert jeweils Einstellungen unterschiedlicher Werte und Zuordnungen. Die Datenfilterung ist sehr gut universell nutzbar, sofern sich sämtliche zu durchsuchenden Daten in einer Tabelle oder Abfrage befinden. Bei immer wiederkehrenden ähnlichen Filterungen führt sie aber nicht so schnell zum Erfolg, wie dies vor allem in Formularen erwünscht wird.

Als praktische Beispiel soll ein Teil einer Mediendatenbank dienen, die in verschiedene Tabellen aufgeteilt ist. Gerade eine Verteilung auf mehrere Tabellen erfordert für ein ordnungsgemäßes Filtern von Daten mehr als nur die Nutzung der internen Funktionen von Base. Schließlich ist die eingebaute Funktionalität auf das Filtern innerhalb einer Tabelle bzw. eines Formulars ohne Unterformular beschränkt.

Der Begriff «Suche» wird in den folgenden Beispielen allerdings so genutzt, wie er auch im Internet üblich ist: Die Eingabe eines beliebigen Suchbegriffs soll zu Ergebnissen führen.

Tabellen



Über **Extras** → **Beziehungen** ist eine Übersicht der Tabellen zusammengefügt, durch die hindurch die Datensuche und Datenfilterung vorgenommen werden soll. Auf eine detaillierte Schilderung der einzelnen Felder soll hier verzichtet werden, da sie für die Suche nicht von Bedeutung ist.

Zentrale Tabelle ist die Tabelle "Medien". Die Tabelle "Kategorien", "Medienart" und "Ort" sind mit der Tabelle über Fremdschlüssel verbunden. Für einen Datensatz aus "Medien" kann es mehrere Datensätze aus "Verfasser" geben. Ebenso kann es für einen Datensatz aus "Verfasser" mehrere

Datensätze aus "Medien" geben. Aus dem Grunde ist über die Tabelle "rel_Medien_Verfasser" eine n:m-Beziehung erstellt worden.

Datenzusammenfassung in einer Ansicht

Damit sämtliche Daten direkt durchsucht werden können müssen sie zuerst in einer Abfrage oder Ansicht zusammengefasst werden. Der Code für eine Ansicht ist gleich dem Code für eine Abfrage. Base greift bei einer Ansicht aber auf die Daten zu, als ob sie direkt aus einer Tabelle kommen. Das läuft schneller.

Zentrale Tabelle ist die Tabelle "Medien":

```
SELECT * FROM "Medien"
```

Damit werden alle Datensätze aus der Tabelle Medien mit allen Feldern wieder gegeben. Die einzelnen Felder müssen nicht benannt werden und werden trotzdem angezeigt.

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie"
   FROM "Medien", "Kategorien"
   WHERE "Medien"."katID" = "Kategorien"."ID"
```

Damit bei zwei Tabellen noch klar ist, aus welcher Tabelle alle Felder angezeigt werden sollen, muss vor «*» die Tabelle verzeichnet sein: "Medien".* .

Aus der Tabelle "Kategorien" werden alle Felder bis auf den Primärschlüssel angezeigt. Das hat mehrere Gründe:

- Der Primärschlüssel aus der Tabelle "Kategorien" ist bereits als "katID" in der Tabelle "Medien" vertreten.
- Der Primärschlüssel aus der Tabelle "Kategorien" hat die gleiche Namensbezeichnung "ID" wie der Primärschlüssel aus der Tabelle "Medien". Während Abfragen so etwas noch in Base verarbeiten gibt die Datenbank eine entsprechende Fehlermeldung aus. "Kategorien"."ID" müsste mit Hilfe einer Aliasbezeichnung abgebildet werden, also z.B. "Kategorien"."ID" AS "KategorienID". Da dieses Feld aber ein Duplikat von "Medien"."katID" wäre ist darauf verzichtet worden.

Der obige Code hat jetzt einen Haken. Es werden nur die Datensätze angezeigt, bei denen "katID" aus der Tabelle "Medien" gleich "ID aus der Tabelle "Kategorien" ist. Ist allerdings bei einem Datensatz in der Tabelle "Medien" das Feld "katID" leer (**`NULL`**), dann ist dieser Datensatz aus der Anzeige ausgeschlossen. Da aber alle Datensätze aus "Medien" angezeigt werden sollen ist hier eine andere Formulierung der Bedingung notwendig:

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie"
   FROM "Medien"
      LEFT JOIN "Kategorien"
      ON "Medien"."katID" = "Kategorien"."ID"
```

Über **`LEFT JOIN`** werden aus der (links von der Verbindung liegenden) Tabelle "Medien" auf jeden Fall alle Datensätze abgebildet. Die Tabellenbezeichnungen folgen nicht direkt aufeinander. Sie werden durch **`LEFT JOIN`** von einander getrennt. Statt **`WHERE`** wird die Beziehung zwischen den Tabellen jetzt mit **`ON`** definiert.

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie",
"Medienart"."ArtKurz", "Medienart"."Medienart", "Medienart"."LeihZeit"
FROM "Medien"
   LEFT JOIN "Kategorien"
   ON "Medien"."katID" = "Kategorien"."ID"
   LEFT JOIN "Medienart"
   ON "Medien"."artID" = "Medienart"."ID"
```

Kommt eine weitere Tabelle dazu, so gilt immer noch: "Medien" ist die Tabelle, in der alle Datensätze liegen. Es werden zu allen Datensätzen aus "Medien" die "Kategorie" angezeigt. Wenn keine

"Kategorie" vermerkt ist wird der Datensatz aus "Medien" trotzdem angezeigt. Ebenso wird zu allen Datensätzen aus "Medien" eine "Medienart" angezeigt. Auch wenn die "Medienart" fehlt wird der Datensatz aus "Medien" weiter angezeigt.

Nur mittels eines **`LEFT JOIN`** (oder, bei umgekehrter Stellung, eines **`RIGHT JOIN`**) lassen sich wirklich alle Datensätze für eine Suche in einem Formular mit Unterformular wirkungsvoll zusammen fassen.

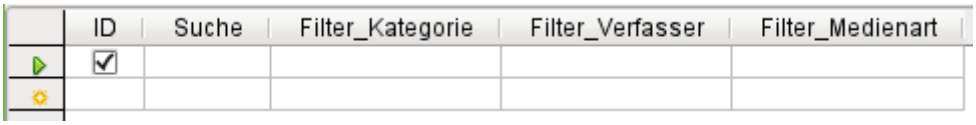
Die gesamte Zusammenfassung aller erforderlichen Daten in der Ansicht "SuchAnsicht" sieht schließlich folgendermaßen aus:

```
SELECT "Medien".* , "Kategorien"."System", "Kategorien"."Kategorie",
"Medienart"."ArtKurz", "Medienart"."Medienart",
"Medienart"."LeihZeit", "Ort"."Ort", "Verfasser"."Verfasser",
"rel_Medien_Verfasser"."VerfSort", "rel_Medien_Verfasser"."vefID" ,
   IFNULL("rel_Medien_Verfasser"."vefID", 0 ) AS "VerfasserID",
   IFNULL("Medien"."katID", 0 ) AS "KategorieID",
   IFNULL("Medien"."artID", 0 ) AS "MedienartID"
FROM "Medien"
   LEFT JOIN "Kategorien"
   ON "Medien"."katID" = "Kategorien"."ID"
   LEFT JOIN "Medienart"
   ON "Medien"."artID" = "Medienart"."ID"
   LEFT JOIN "Ort"
   ON "Medien"."ortID" = "Ort"."ID"
   LEFT JOIN "rel_Medien_Verfasser"
   ON "rel_Medien_Verfasser"."medID" = "Medien"."ID"
   LEFT JOIN "Verfasser"
   ON "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
```

Die mit einem Alias versehenen letzten drei Felder dienen dazu, dass für die anschließende Filterung nur Felder gefiltert werden, in denen auf jeden Fall ein Eintrag vorliegt. Ohne diese Zusätze könnte bei einer Filterung nicht mehr auf alle Datensätze zugegriffen werden. Siehe zu dieser Problematik die Abfrage «Filter_Form_Start».

Erstellung einer Filter-Tabelle

Die Filterung erfolgt, indem in eine Tabelle mit nur einem Datensatz über das Formular ein Wert eingetragen wird. Dieser Wert wird über eine Abfrage ausgewertet, so dass alle Datensätze mit diesem entsprechenden Wert angezeigt werden können.



	ID	Suche	Filter_Kategorie	Filter_Verfasser	Filter_Medienart
▶	☑				
✱					

Die Tabelle "Filter" hat als Primärschlüssel das Feld "ID". Der Primärschlüssel ist hier ein einfaches Ja/Nein-Feld. Der erste Wert für dieses Schlüsselfeld ist bereits gesetzt.

Daneben sind Felder für einen Suchbegriff in Textform sowie für verschiedene Fremdschlüsselfilter (Datenformat INTEGER) vorgesehen. In dem Feld "Filter_Kategorie" z. B. wird also nur ein Zahlenwert abgespeichert, nach dem dann die Datensätze durchsucht werden sollen.

Hinweis	Mit einer festen Filtertabelle lässt sich nur arbeiten, wenn es sich bei der Datenbank um die interne Datenbank handelt. Sonst würden die Filterwerte gleichzeitig bei anderen Nutzern erscheinen. Für Serverdatenbanken wie MySQL/MariaDB kann stattdessen mit einer temporären Tabelle gearbeitet werden. Eine temporäre Tabelle ist immer auf die momentane Datenbankverbindung angemeldet. Die temporäre Tabelle wird anderen Nutzern nicht angezeigt.

Ansichten für eine Standardfilterung mit Subformular

Innerhalb von Base gibt es natürlich auch die Möglichkeit, mit Hilfe des Dialogs der Standardfilterung Datensätze zu filtern. Diese Filterung beschränkt sich allerdings auf die dem jeweiligen Formular zugrundeliegende Datenquelle. Ein Unterformular wird dabei nicht berücksichtigt.

Mit Hilfe einer speziellen Formularkonstruktion und der "SuchAnsicht" kann allerdings der Standardfilter auch scheinbar über mehrere Formulare hinweg genutzt werden. Das wird später bei der Konstruktion der Formulare noch genauer erklärt.

Nachteil der "SuchAnsicht" ist allerdings, dass einzelne Medien mehrfach in der Ansicht vorkommen, weil mehrere Verfasser den Medien zugeordnet sind. Dadurch erscheinen später scheinbar mehr Datensätze, als die Tabelle "Medien" zu dem entsprechenden Suchbegriff überhaupt hat. Die Verfasser müssen deshalb in einem Feld zusammen gruppiert werden.

Prinzipiell wäre so etwas mit einer Gruppierungsfunktion der Datenbank lösbar. Nur kennt die eingebaute HSQLDB keine Funktion wie **`GROUP_CONCAT`**() oder **`LIST`**(). Deshalb ist eine Staffelung mehrere Ansichten erforderlich, damit diese Gruppierung gelingt. Nachteil dabei ist leider, dass diese Gruppierung bei entsprechend großen Datenbeständen nicht gerade schnell vonstatten geht. **Die folgenden Ansichten sind nur für die korrekte Bedienung des Standardfilters mit einem Subformular erforderlich.**

1. Ansicht: FilterStartAnsicht

```
SELECT "ID", "System", "Kategorie", "Titel", "Jahr", "ArtKurz",
"Medienart", "LeihZeit", "Ort", (
   SELECT COUNT( "ID" ) FROM "SuchAnsicht"
   WHERE "ID" = "a"."ID" AND "vefID" <= "a"."vefID"
) AS "GruppenNr", "Verfasser"
FROM "SuchAnsicht" AS "a"
```

Diese Ansicht greift auf die vorher erstelle "SuchAnsicht" zu. So wird vermieden, dass gleiche Abfrageinhalte mehrmals hintereinander die Datenbank belasten. Neben den für die spätere Filterung vorgesehenen Feldern wird in einer korrelierenden Unterabfrage eine Nummerierung erstellt. **`SELECT COUNT( "ID" ) FROM "SuchAnsicht" WHERE "ID" = "a"."ID" AND "vefID" <= "a"."vefID"`** zählt die Einträge im Feld "ID" (weil dieses Feld auf jeden Fall einen Eintrag hat). Die Unterabfrage liest dabei den Wert für "ID" aus dem aktuellen Datensatz der Hauptabfrage aus: **`"ID" = "a"."ID"`**. Außerdem sieht sie noch nach, ob der Fremdschlüsseleintrag für den Verfasser in der Unterabfrage kleiner oder gleich dem Eintrag im aktuellen Datensatz ist: **`"vefID" <= "a"."vefID"`**. Dadurch werden die Verfasser für jedes Medium, repräsentiert durch "ID", durchnummeriert. Diese Nummerierung ist notwendig, damit eine entsprechende Gruppierung der unterschiedlichen Verfasser für ein Medium in einem einzigen Feld erfolgen kann.

2. Ansicht: StandardfilterAnsicht

```
SELECT DISTINCT "ID", "System", "Kategorie", "Titel", "Jahr",
"ArtKurz", "Medienart", "LeihZeit", "Ort",
   ( SELECT "Verfasser" FROM "FilterStartAnsicht"
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 1 ) ||
   IFNULL( '; ' || ( SELECT "Verfasser" FROM "FilterStartAnsicht"
```

```
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 2 ), '' ) ||
   IFNULL( '; ' || ( SELECT "Verfasser" FROM "FilterStartAnsicht"
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 3 ), '' ) ||
   IFNULL( '; ' || ( SELECT "Verfasser" FROM "FilterStartAnsicht"
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 4 ), '' ) ||
   IFNULL( '; ' || ( SELECT "Verfasser" FROM "FilterStartAnsicht"
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 5 ), '' ) ||
   IFNULL( '; ' || ( SELECT "Verfasser" FROM "FilterStartAnsicht"
      WHERE "ID" = "a"."ID" AND "GruppenNr" = 6 ), '' )
   AS "Verfasser"
FROM "FilterStartAnsicht" AS "a"
```

Basis der zweiten Ansicht "StandardfilterAnsicht" ist die vorhergehende Ansicht "FilterStartAnsicht". Es werden in dieser Ansicht nur die Datensätze ausgegeben, die unterschiedliche Werte haben. Dies geht aus dem Zusatz **`DISTINCT`** hervor. Aus dem Feld "Verfasser" der "FilterStartAnsicht" werden zuerst die Inhalte herausgesucht, bei denen **`"GruppenNr" = 1`** ist. Die "Verfasser" pro "ID" wurden ja in der vorhergehenden Abfrage durchnummeriert. Die Gruppennummer 1 hat jetzt der Datensatz aus "Verfasser", der den kleinsten Primärschlüssel in der Tabelle "Verfasser" hat.

An den ersten ermittelten Wert für "Verfasser" werden die weiteren Werte für "Verfasser" angehängt: **`||`** . Sobald allerdings eine der Abfragen leer ist würde das ohne zusätzlichen Eingriff ein leeres Feld für den "Verfasser" geben. Deshalb werden die weiteren Unterabfragen mit **`IFNULL`** abgesichert. Wenn die Unterabfrage z.B. für die **`"GruppenNr" = 2`** leer bleibt wird stattdessen einfach ein leerer Text eingesetzt.

Abfragen

Die meisten der folgenden Abfragen werden als Datengrundlage für die Formulare genutzt. Abfragen, die als Datengrundlage für Formulare dienen, sollten am besten editierbar sein. Dies wurde bei allen Abfragen berücksichtigt.

Eine Abfrage ist dann editierbar, wenn sie die Primärschlüssel aller Tabellen enthält, auf die sich die Abfrage bezieht.

Filter_Form_Start

Diese Abfrage ist lediglich in die Beispieldatenbank aufgenommen worden, um ein eventuell häufig auftauchendes Problem bei dem Einsatz von Filtern auf zu zeigen.

```
SELECT *
FROM "Medien"
WHERE "katID" =
   IFNULL(
   ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
   "katID" )
```

`SELECT * FROM "Medien"`: Zeige alle Datensätze aus der Tabelle "Medien" an.

`WHERE "katID" =`: Bei denen das Feld "katID" gleich dem folgenden Wert ist. Diese Bedingung schließt so bereits aus, dass das Feld "katID" **`NULL`** sein darf.

`IFNULL( <Bedingung> , "katID")`: Wenn die Bedingung keinen Datensatz zurück gibt, also **`NULL`** ist, dann soll stattdessen das Feld "katID" hinter dem Gleichheitszeichen erscheinen. Es würde also bei **`NULL`** lauten: **`WHERE "katID" = "katID"`**. Ist die Bedingung also leer, dann werden alle Datensätze angezeigt, bei denen in "katID" ein Eintrag vorliegt.

Die nachgefragte Bedingung ist eine Abfrage an das Feld "Filter_Kategorie" aus der Tabelle "Filter", bei der der Wert für den Primärschlüssel "ID" wahr (**`TRUE`**) ist. Die Filter-Tabelle wurde ja mit

einem Primärschlüssel versehen, der aus einem Ja/Nein Feld (auch: wahr/falsch oder TRUE/FALSE oder 1/0) besteht. Mit dieser Unterabfrage wird höchstens ein Wert wieder gegeben. Da diese Unterabfrage lediglich in der Bedingung für die Datenauswahl erscheint ändert sie nichts daran, dass die Abfrage insgesamt editierbar bleibt.

Filter_Form

Die vorherige Abfrage hat das Problem, dass durch die Filterung von vornherein alle Datensätze ausgenommen sind, bei denen das Feld "katID" **NULL** ist. Deswegen muss der Code so ergänzt werden, dass in dem Vergleichsfeld aus der Abfrage immer ein Wert erscheint. Statt also bei einem leeren Feld "katID" keine Ausgabe zu erzeugen wird stattdessen ein Wert ausgegeben, der als "katID" nicht erscheint, da er nicht als Primärschlüssel in der Tabelle "Kategorien" vorgesehen ist.

```
SELECT "Medien".*, IFNULL( "katID", 0 ) AS "kat"
FROM "Medien" WHERE "kat" =
   IFNULL(
   ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
   "kat" )
```

Das Feld "katID" nimmt in der Beispieldatenbank mindestens den Wert '1' an. Durch die Zuweisung **IFNULL("katID", 0) AS "kat"** wird einem neuen Feld mit der Aliasbezeichnug "kat" zuerst einmal immer der Wert von "katID" zugewiesen. Ist "katID" leer (**NULL**), dann wird in "kat" stattdessen '0' angezeigt. "kat" hat also im Gegensatz zu "katID" immer einen Wert vorzuweisen.

Mit diesem Feld "kat" wird jetzt der Vergleich durchgeführt. Ergibt die Unterabfrage **SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE** keinen Wert, dann wird stattdessen "kat" mit "kat" verglichen. Alle Datensätze der Tabelle "Medien" (sowie außerdem "kat") werden angezeigt. Auch die Datensätze, bei denen "katID" NULL ist.

Diese Abfrage kann also komplett so in einem Formular genutzt werden, in dem nur nach der Kategorie gefiltert werden soll. Durch die Filterung passiert es nicht, dass von vornherein bestimmte Datensätze nicht angezeigt würden.

Filter_Form_Subform

Wird nicht nur ein einfaches Formular durchsucht, sondern z.B. der Inhalt eines Subformulars, so kann nicht direkt auf die Tabelle "Medien" des Hauptformulars Bezug genommen werden. Allerdings soll die Filterung des Unterformulars nicht, wie bei den eingebauten Filtern, nur das Unterformular filtern und immer noch alle Datensätze des Hauptformulars anzeigen. Stattdessen sollen die Datensätze ausgeschlossen werden, bei denen der gewünschte Begriff im Unterformular nicht vorhanden ist.

Für diese Art der Filterung wird die Ansicht "SuchAnsicht" durchsucht. Es wird in dem Beispiel nach einem Verfasser, genauer nach dem Fremdschlüssel des Verfassers "vefID" gesucht. Der Fremdschlüssel steht nicht in der Tabelle "Medien", da das Verhältnis von "Verfasser" zu "Medien" ein n:m-Verhältnis ist.

```
SELECT * FROM "Medien"
WHERE "ID" IN (
   SELECT DISTINCT "ID" FROM "SuchAnsicht" WHERE "VerfasserID" =
   IFNULL(
      ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
   "VerfasserID" )
   )
```

Durch die Abfrage werden alle Werte der Tabelle "Medien" angezeigt, auf die die Bedingung zutrifft: **SELECT * FROM "Medien"** .

Mit der Bedingung **WHERE "ID" IN ( )** kann über eine Unterabfrage der Inhalt in den Klammern auch deutlich mehr als nur einen Datensatz anzeigen. Nur die Bedingung **IN ( )** gilt für ganze Datengruppen, aus denen heraus der passende Wert gesucht wird.

Innerhalb der Klammern wird über **SELECT DISTINCT "ID" FROM "SuchAnsicht"** eine Gruppe von Primärschlüsselwerten der Tabelle "Medien" angegeben. Der Zusatz **DISTINCT** führt dazu, dass in diesem Fall kein Wert des Feldes "ID" doppelt erscheint.

Schließlich wird die Bedingung wieder, wie in der Abfrage «Filter_Form», in diesem Fall aber für das Feld "VerfasserID", formuliert. Das Feld "VerfasserID" ist in der Suchansicht bereits nach dem gleichen Verfahren erstellt worden, nach dem in der Abfrage «FilterForm" das Feld "kat" erstellt wurde: **IFNULL("rel_Medien_Verfasser"."vefID", 0) AS "VerfasserID"** . Damit ist sicher gestellt, dass bei einem nicht ausgewählten Verfasser auch die Datensätze angezeigt werden, die eventuell mit gar keinem Verfasser verbunden sind.

Filter_Form_Subform_3Filter

Mit Hilfe des gleichen Verfahrens wie bei "Filter_Form_Subform" können auch gleichzeitig mehrere Felder gefiltert werden.

```
SELECT * FROM "Medien"
WHERE "ID" IN (
   SELECT DISTINCT "ID" FROM "SuchAnsicht" WHERE "VerfasserID" =
      IFNULL(
      ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
      "VerfasserID" )
      AND "MedienartID" = IFNULL(
      ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" = TRUE ),
      "MedienartID" )
      AND "KategorieID" = IFNULL(
      ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
      "KategorieID" )
   )
```

Wieder werden alle Felder in "SuchAnsicht" gelistet, die den Bedingungen entsprechen. Dabei werden die einzelnen Filter mit **AND** verknüpft. Es sollen also nur die Felder angezeigt werden, die wirklich allen eingestellten Filtern entsprechen. Damit wird die engste Auswahl getroffen. Hier könnte natürlich auch mit einem **OR** die Auswahl so dargestellt werden, dass entweder der eine oder der andere Wert enthalten sein muss. Nur würden in diesem Falle nicht ausgewählte Filter dazu führen, dass auf jeden Fall alle Datensätze aus "Medien" angezeigt werden.

Listenfeld_Kategorie_ohne_Eintrag

Sollen mit den Listenfeldern gegebenenfalls auch die Felder gefiltert werden, die gar keinen Eintrag haben (um z.B. diesen Fehler zu beheben …), so muss das Listenfeld neben den Fremdschlüsselwerten der eigentlichen Tabelle auch den Eintrag für gar keinen Fremdschlüsselwert aufzeigen. In der Abfrage "Filter_Form" wurde über den Alias "kat" Feldern ohne Fremdschlüssel der Wert '0' zugewiesen: **IFNULL("katID", 0) AS "kat"** . Dieser Wert muss also bei der Filterung durch das Listenfeld gegebenenfalls auch in die Tabelle "Filter" eingetragen werden können.

```
SELECT "Kat", "ID"
FROM (
   SELECT '1' AS "C",
      'ohne Eintrag' AS "Kat",
      0 AS "ID"
      FROM "Kategorien"
   UNION
   SELECT "Kategorien"."System",
```

```
LEFT( "Kategorien"."System" || SPACE( 7 ), 7 ) || ' - ' ||
   "Kategorien"."Kategorie" AS "Kat",
"Kategorien"."ID"
FROM "Kategorien", "Medien"
WHERE "Kategorien"."ID" = "Medien"."katID")
```

Der Start besteht, wie sonst bei Listenfelder üblich, aus der Aufzählung von zwei Felder. Das zuerst ermittelte Feld "Kat" zeigt den Text, der später im Listenfeld zu sehen ist. Das Feld "ID" wird nicht angezeigt. Es stellt die Verbindung zum Formularfeld und dem Feld der Tabelle bzw. Abfrage her, das dem Formularfeld zugrunde liegt.

In der Bedingung der Abfrage werden zwei Abfragen miteinander durch **`UNION`** verbunden. Diese Verbindung funktioniert nur, wenn die Abfragen sich auf Tabellen mit gleichen Feldtypen bezieht. Es kann auch nicht eine Tabelle mit 4 Feldern mit einer anderen mit 3 Feldern verbunden werden.

Standardmäßig werden keine doppelten Werte angezeigt. Dies entspricht dem Zusatz **`DISTINCT`**, wie er in den vorhergehenden Abfragen gebraucht wurde.

Auch eine Sortierung ist nicht möglich. Stattdessen wird fest nach dem ersten Feld der Abfragen/Tabellen sortiert.

Die erste Abfrage **`SELECT '1' AS "C", 'ohne Eintrag' AS "Kat", 0 AS "ID" FROM "Kategorien"`** schreibt in das Sortierfeld eine '1' und weist diesem Feld einen (beliebigen) Alias "C" zu. Mit dem Wert '1' wird sicher gestellt, dass in der späteren Sortierung dieser Eintrag ganz oben steht. Sämtliche Einträge aus "Kategorien"."System" beginnen nämlich mit Buchstaben, werden also nach einer Zahl eingeordnet.

Im Listenfeld erscheint also zuerst «ohne Eintrag». Wird «ohne Eintrag» ausgewählt, so wird der Wert '0' an das darunterliegende Feld der Tabelle weiter gegeben.

Die zweite Abfrage sucht über **`WHERE "Kategorien"."ID" = "Medien"."katID"`** nur die Kategorien heraus, die auch tatsächlich in der Tabelle "Medien" eingetragen sind. Dies ist hier notwendig, da die Kategorisierung der "Medien" einer allgemeinen Vorgabe für Bibliotheken entspricht und viele Kategorien bisher gar nicht genutzt werden.

Das erste Feld stellt das Systematikkürzel dar: **`"Kategorien"."System"`**. Nach diesem Feld wird durch **`DISTINCT`** sortiert.

Das zweite Feld kombiniert die Systematik mit der Kategorie: **`LEFT( "Kategorien"."System" || SPACE( 7 ), 7 ) || ' - ' || "Kategorien"."Kategorie" AS "Kat"`** . Da die Übersicht in dem Listenfeld tabellarisch dargestellt werden soll, wird die Länge des Eintrages für "Kategorien"."System" auf 7 Zeichen festgelegt. Dies entspricht dem längsten Eintrag in diesem Feld. Gegebenenfalls wird also über SPACE(7) mit maximal 7 Leerzeichen aufgefüllt. Mit der Darstellung von "Kategorien"."System" wird über einen Bindestrich «-» die Beschreibung "Kategorien"."Kategorie" verbunden. Die **`||`** dienen hier als die Elemente, die die Verbindung von beliebig vielen Feldern zu einem einzigen Feld gestatten.[1]

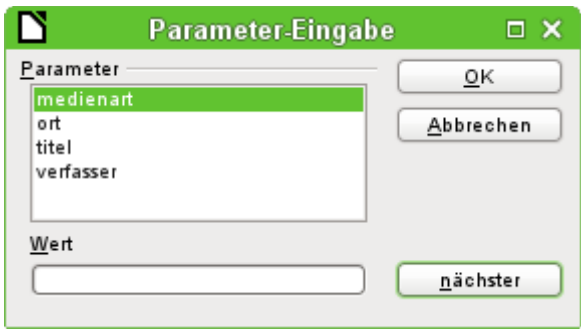
Im dritten Feld der zweiten Abfrage wird lediglich der Primärschlüssel der Tabelle "Kategorien" abgefragt, der zu dem jeweiligen Datensatz passt.

Die Hauptabfrage **`SELECT "Kat", "ID"`** fragt nun aus dieser Zusammensetzung der beiden Unterabfragen das zweite und das dritte Feld ab. Das erste Feld der Unterabfragen dient lediglich der Sortierung.

Parametersuche_Medien

Die Parametersuche hat den Vorteil, dass hier auch direkt aus der Abfrage heraus über die Eingabe von Parametern ein Suchergebnis gefiltert werden kann.

1 Der senkrechte Strich (engl.: vertical bar, in der Programmiersprache auch «pipe» genannt) wird in der EDV als «Verkettungszeichen» bezeichnet.



Wird die Parametersuche direkt aufgerufen, so erfolgt eine Abfrage der Parameter über einen Dialog.

Wird die «Parametersuche_Medien» hingegen aus einem Formular heraus angesprochen, so erfolgt das Auslesen der weiter zu gebenden Werte wieder aus der Tabelle "Filter".

```
SELECT *
FROM "Medien"
WHERE "ID" IN (
   SELECT "ID"
   FROM "SuchAnsicht"
   WHERE
      IFNULL( LOWER ( "Titel" ), '' ) LIKE
         IFNULL( LOWER ( :titel ), '' ) || '%' AND
      IFNULL( LOWER ( "Medienart" ), '' ) LIKE
         IFNULL( LOWER ( :medienart ), '' ) || '%' AND
      IFNULL( LOWER ( "Ort" ), '' ) LIKE
         IFNULL( LOWER ( :ort ), '' ) || '%' AND
      IFNULL( LOWER ( "Verfasser" ), '' ) LIKE
         IFNULL( LOWER ( :verfasser ), '' ) || '%'
   )
```

Wird diese Abfrage geöffnet, so erscheint erst einmal ein Dialog, in dem die Begriffe «titel», «medienart» usw. abgefragt werden. Durch den Doppelpunkt vor diesen Begriffen liest Base diese Begriffe als Parameter aus, die zuerst einmal gefüllt werden müssen.

Es sollen alle Datensätze aus der Tabelle "Medien" aufgezeigt werden, bei denen das Feld "ID" in den Feldern "ID" der Unterabfrage enthalten ist. In der Unterabfrage wird die "SuchAnsicht" nach entsprechenden Begriffen durchsucht. Dies ist erforderlich, da die Tabelle "Medien" die entsprechenden Angaben nicht enthält. Die Tabelle "Medien" ist aber zur Ansteuerung des entsprechenden Formulars notwendig, wenn dort eine Eingabe möglich sein soll.

Ist tatsächlich ein Inhalt des Feldes "Titel" **NULL**, so soll dort stattdessen ein leerer Text als Inhalt des Feldes mit der Bedingung verknüpft werden. Gleiches gilt für die anderen Felder.

Parametereingaben sind bis LO Version 4.4 nie **NULL**, wenn sie über den Dialog eingegeben werden. Der Dialog gibt stattdessen einen leeren Text weiter. Die Abfrage enthält dennoch einen entsprechenden Passus zur Überprüfung der Parameter, da Parameter auch über Formulare aus einer Tabelle heraus an die Abfrage weiter gegeben werden können. Und leere Felder sind dort in der Regel eben **NULL**.

Seit der Version LO 4.4 gestaltet sich die obige Abfrage entsprechend einfacher, da Abfrage und Formular jetzt identisch funktionieren und auch die Parametereingabe in der Abfrage bei leerem Feld **NULL** wiedergibt:

```
SELECT *
FROM "Medien"
WHERE "ID" IN (
   SELECT "ID"
```

```
FROM "SuchAnsicht"
WHERE
   (LOWER ( "Titel" ) LIKE LOWER ( :titel ) || '%'
      OR :titel IS NULL) AND
   (LOWER ( "Medienart" ) LIKE LOWER ( :medienart ) || '%'
      OR :medienart IS NULL) AND
   (LOWER ( "Ort" ) LIKE LOWER ( :ort ) || '%'
      OR :ort IS NULL) AND
   (LOWER ( "Verfasser" ) LIKE LOWER ( :verfasser ) || '%'
      OR :verfasser IS NULL)
)
```

Suche_Form

Bei der Filterung wurden über Listenfelder nach festen Fremdschlüsseln Daten ausgefiltert. Die Suche in dieser Beispieldatenbank erlaubt die Eingabe eines beliebigen Begriffes in ein Textfeld. Dabei soll die Suche ähnlich funktionieren, wie es viele Nutzer von Suchmaschinen im Internet gewohnt sind.

Der Suchbegriff wird wie auch die Filter-Fremdschlüssel in der Datei "Filter" gespeichert und von dort aus für die Abfrage "Suche_Form" ausgelesen.

```
SELECT *
FROM "Medien"
WHERE LOWER ( "Titel" ) LIKE
   IFNULL( '%' || LOWER (
      ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
      ) || '%',
      LOWER ( "Titel" ) )
```

Alle Datensätze aus der Tabelle "Medien" sollen angezeigt werden, die den Kriterien entsprechen: **`SELECT * FROM "Medien"`** . Die Suche wird nur für das Feld "Titel" durchgeführt, da in der Beispieltabelle nur im Feld "Titel" Text steht. Es ist allerdings genauso gut möglich, zusätzlich andere Felder, auch Zahlenfelder wie z.B. "Medien"."Jahr" oder auch "Medien"."ID" mit dem gleichen Begriff auf einmal zu filtern. Dies wird in der Abfrage «Suche_Form_Subform» gezeigt.

`LOWER ("Titel")` wird mit dem Suchbegriff verglichen. **`LOWER`** bedeutet hier lediglich, dass der Inhalt des Feldes "Titel" komplett in Kleinbuchstaben ausgelesen wird. Damit soll erreicht werden, dass Suchbegriffe unabhängig von Groß- und Kleinschreibung gefunden werden.

Die Bedingung **`LIKE`** durchsucht den Inhalt eines Feldes. Mit dem Zusatz «%» sind beliebig viele Zeichen gemeint. Vor dem Suchbegriff sowie nach dem Suchbegriff dürfen also beliebig viele Zeichen stehen: **`'%' || LOWER ( ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'`** . Aus dem Feld "Suche" der Tabelle "Filter" wird der Begriff ausgelesen und gleichzeitig in Kleinbuchstaben umgewandelt. Ist kein Begriff dort eingetragen, so ist das Feld **`NULL`**. Damit wird auch die Zusammenfügung der Zeichen «%» mit **`NULL`** zu **`NULL`**. Sobald also kein Suchbegriff da ist soll statt der Auswertung des Eintrages in der Tabelle "Filter" mit **`LOWER ("Titel")`** verglichen werden. Da alle Datensätze in der Tabelle "Medien" mit einem Titel versehen wurden werden so alle Datensätze angezeigt, wenn in der Tabelle "Filter" kein Suchbegriff eingetragen wurde.

Wie die Abfrage "Filter_Form" funktioniert auch die Abfrage "Suche_Form" so nur in einem Formular, nicht auch im Unterformular.

Suche_Form_Subform

Der Ansatz, wie hier nicht nur im Hauptformular, sondern auch im Unterformular gesucht werden kann, entspricht dem Ansatz der Abfrage «Filter_Form_Subform».

```
SELECT *
FROM "Medien"
WHERE "ID" IN (
   SELECT DISTINCT "ID"
   FROM "SuchAnsicht"
   WHERE
      LOWER ( "Titel" ) LIKE
         IFNULL( '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
         WHERE "ID" = TRUE ) ) || '%', LOWER ( "Titel" ) )
      OR LOWER ( "Kategorie" ) LIKE
         '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
         WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Medienart" ) LIKE
         '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
         WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Ort" ) LIKE
         '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
         WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Verfasser" ) LIKE
         '%' || LOWER ( ( SELECT "Suche" FROM "Filter"
         WHERE "ID" = TRUE ) ) || '%'
)
```

Mit **`SELECT * FROM "Medien" WHERE "ID" IN (SELCET DISTINCT "ID" FROM "SuchAnsicht WHERE … )`** werden alle Schlüsselwerte "ID" über die "SuchAnsicht" ermittelt. Aus der Tabelle "Medien" werden zu diesen Schlüsselwerten die Datensätze dargestellt.

Das Feld "Titel" wird genauso abgeglichen wie in der Abfrage «Suche_Form». Wenn also keine Eingabe in dem Feld "Suche" aus der Tabelle "Filter" steht, dann wird einfach **`LOWER ("Titel")`** mit **`LOWER ("Titel")`** verglichen. Dadurch werden alle Datensätze angezeigt.

Alle anderen durchsuchten Felder werden in der Bedingung mit **`OR`** verbunden. Das Suchwort soll also entweder in dem Feld "Titel" oder in dem Feld "Kategorie" oder in dem Feld "Medienart" usw. enthalten sein. Würden die Felder mit **`AND`** verbunden, so würden die Ergebnisse unnötig eingeschränkt.

Da die Bedingungen mit **`OR`** verbunden sind ist es außerdem nicht erforderlich, mehr als ein Feld so zu gestalten, dass bei einem leeren Feld "Suche" eben alle Datensätze angezeigt werden.

Suche_Filter_Form_Subform_3Filter

```
SELECT *
FROM "Medien"
WHERE "ID" IN (
   SELECT DISTINCT "ID"
   FROM "SuchAnsicht"
   WHERE
      "VerfasserID" = IFNULL(
         ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
         "VerfasserID" )
      AND "MedienartID" = IFNULL(
         ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" = TRUE ),
         "MedienartID" )
      AND "KategorieID" = IFNULL(
         ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
         "KategorieID" )
      AND (
         LOWER ( "Titel" ) LIKE IFNULL( '%' || LOWER (
```

```
            ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
            ) || '%', LOWER ( "Titel" ) )
         OR LOWER ( "Kategorie" ) LIKE '%' || LOWER (
            ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
            ) || '%'
         OR LOWER ( "Medienart" ) LIKE '%' || LOWER (
            ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
            ) || '%'
         OR LOWER ( "Ort" ) LIKE '%' || LOWER (
            ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
            ) || '%'
         OR LOWER ( "Verfasser" ) LIKE '%' || LOWER (
            ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE )
            ) || '%'
      )
)
```

Mit dieser Abfrage werden schließlich alle Filterfelder und das Suchfeld kombiniert. Sie ist lediglich eine Kombination der Abfrage Filter_Form_Subform_3Filter mit der Abfrage Suche_Form_Subform. Die Bedingung der Abfrage Suche_Form_Subform ist einfach an die Bedingung für die Filterung angehängt worden. Zu beachten ist lediglich, dass die Suchbedingung wegen der **OR**-Verknüpfung **innerhalb einer Klammer** angefügt wurde. Sonst würde durch die Suchbedingung jede mögliche Filterung ausgeschlossen. Die Suchbedingung ist schließlich so konstruiert, dass bei einer Eingabe entweder in dem Feld "Titel" oder in dem Feld "Kategorie" oder dem Feld "Medienart" oder dem Feld "Ort" oder dem Feld "Verfasser" ein entsprechender Wert liegen darf. Dadurch würden die Einschränkungen der Filter ohne entsprechende Klammerung teilweise wieder aufgehoben.

Formulare

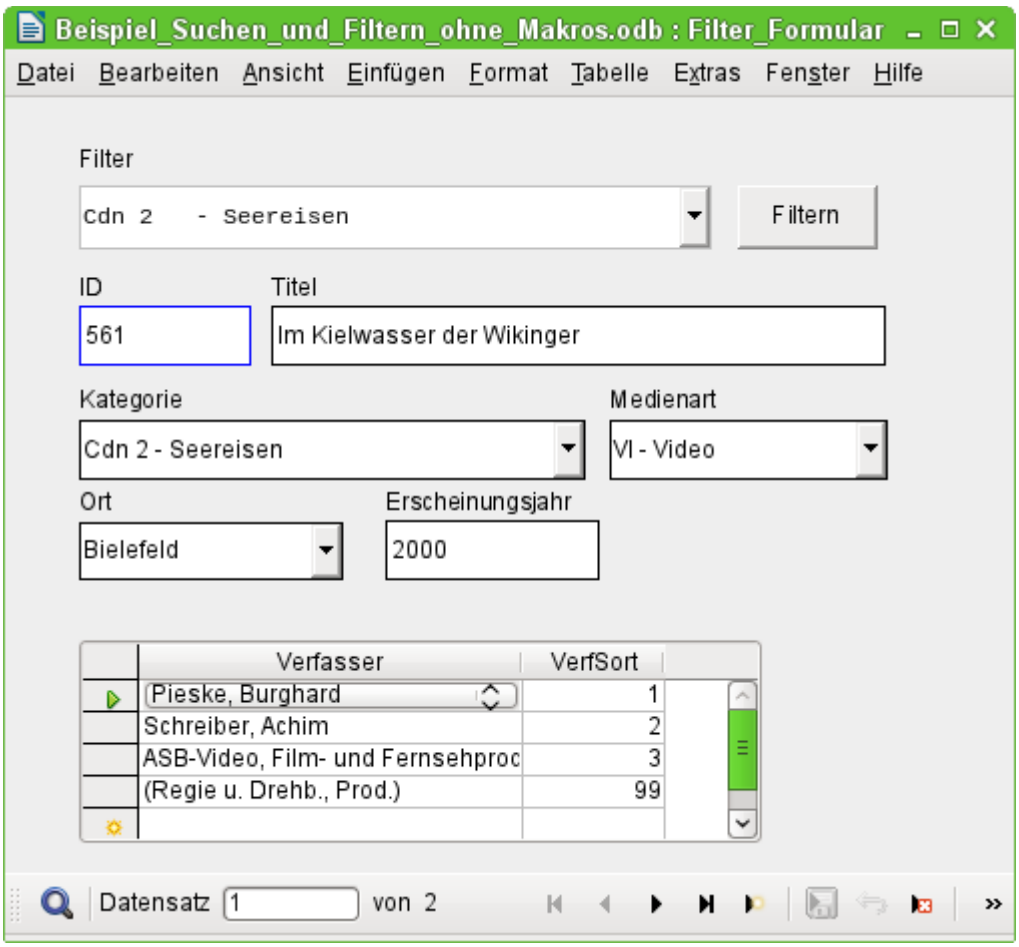
Bei den Formularen wird zuerst der zu ermittelnde Wert ausgesucht. Danach wird die Auswahl durch einen Button bestätigt.

Im darunter liegenden Formularteil wird das Filter- bzw. Suchergebnis angezeigt. Die Fundstellen in dem Formular werden nicht weiter markiert.

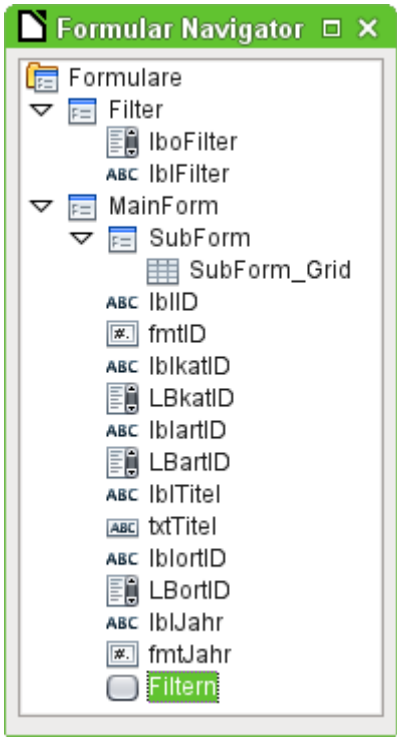
Filter_Formular

Äußerlich unterscheiden sich die Formulare «Filter_Formular», «Filter_Formular_Nullwerte» und «Filter_Formular_Subformular» nicht.

Über ein Listenfeld wird ein Wert ausgesucht. Der Button Filtern wird betätigt. Das Formular wird auf die ermittelten Datensätze eingestellt.



In diesem Beispiel wurde nach der Kategorie «Cdn 2» gesucht, die für den Begriff «Seereisen» steht. In der Datenbank gibt es insgesamt zwei Datensätze, die mit dieser Kategorie verzeichnet sind. Dies ist aus der Navigationsleiste ersichtlich.

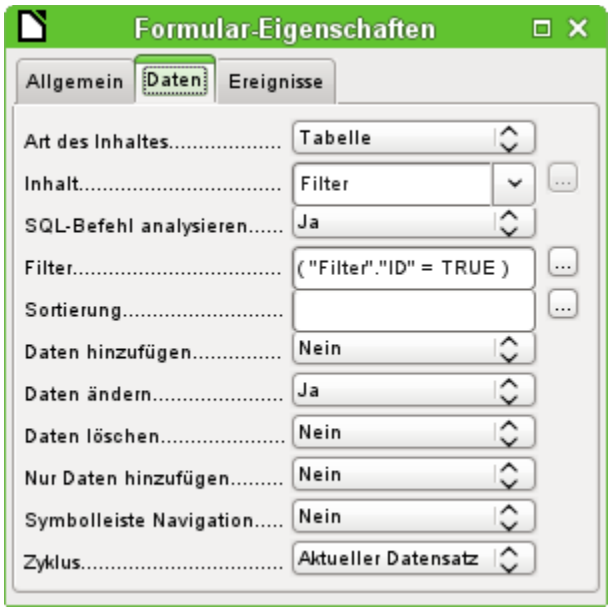


Der Formular Navigator gibt Aufschluss darüber, wie das Formular konstruiert ist. Die Konstruktion ist prinzipiell bei allen anderen Formularen dieser Beispieldatenbank gleich.

Es gibt zwei Formulare, die gleichberechtigt nebeneinander stehen.

Das Formular «Filter» enthält nur ein Listenfeld und das dazugehörige Beschriftungsfeld. Datengrundlage für dieses Formular ist die Tabelle "Filter". Mit dem Listenfeld ist das Feld "Filter_Kategorie" verbunden.

Das Formular «Filter» ist nicht für die Eingabe neuer Werte vorgesehen. Mit einem rechten Mausklick auf den Eintrag «Filter» im Formular Navigator und dem Menüpunkt «Eigenschaften» werden die Eigenschaften des Formulars sichtbar:



Das Formular zeigt nur den Datensatz an, für den aus der Tabelle "Filter" der Wert im Feld "ID" **TRUE** ist. Es werden keine Daten hinzugefügt, da dieser Datensatz ja bereits existiert. Es werden auch keine Daten gelöscht. Beides scheint hier etwas irritierend. Mit «Daten» ist hier aber der gesamte Datensatz gemeint. Einzelne Felder können dagegen geändert und auch geleert werden.

Eine Symbolleiste Navigation ist für dieses Formular nicht notwendig, da es sich immer im Datensatz 1 von 1 bewegt.

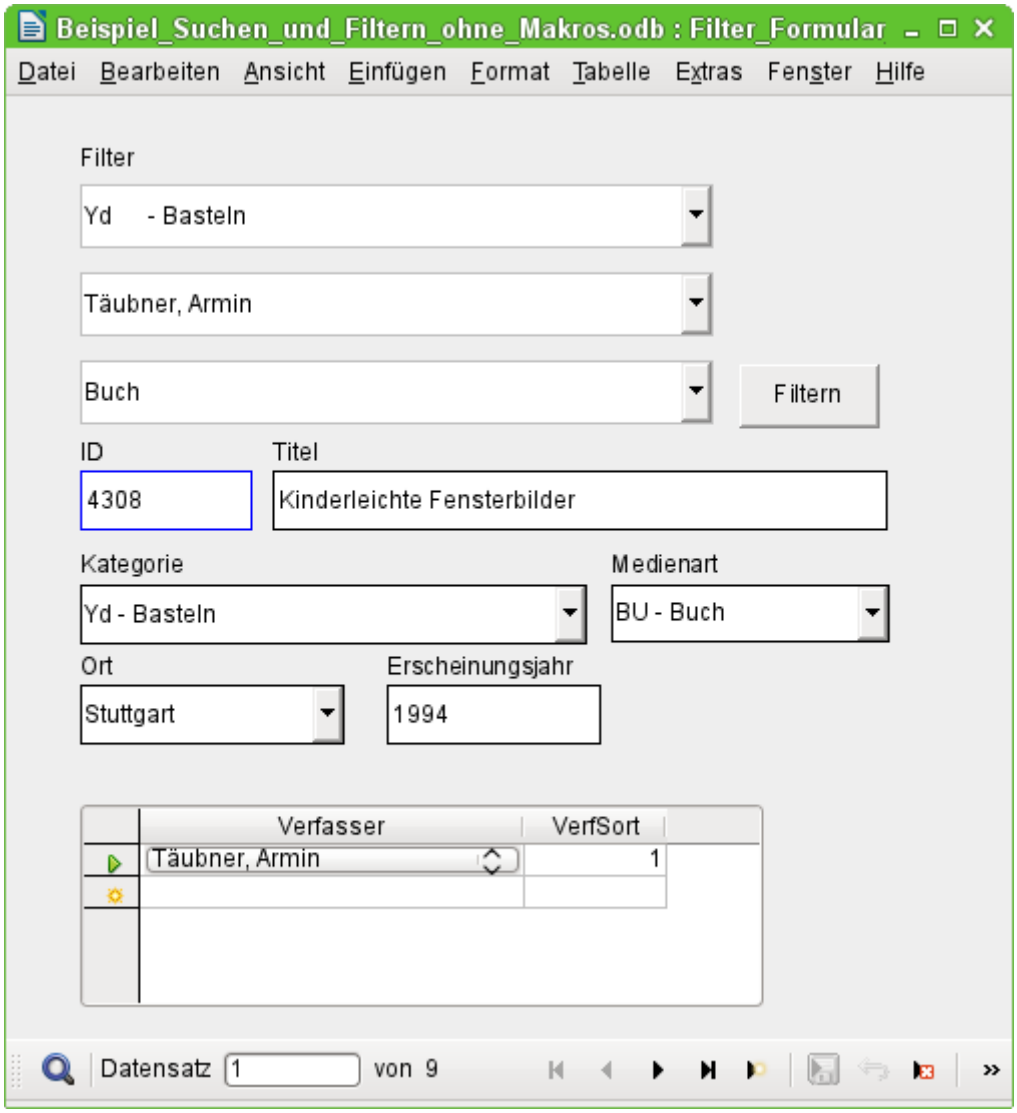
Das zweite Formular «MainForm» steht unabhängig neben dem Formular «Filter». Dieses Formular hat als Datenbasis eine Abfrage. Für das Gesamtformular «Filter_Formular» ist dies die Abfrage «Filter_Form». Diese Abfrage gibt die Datensätze nach dem im Nebenformular «Filter» eingetragenen Wert wieder.

Wichtigstes Element in dem Formular «MainForm» ist der Button «Filtern». Dieser Button ist mit der Aktion «Formular aktualisieren» verbunden. Er aktualisiert also die Anzeige des Formulars «MainForm» und des damit verknüpften Unterformulars «SubForm». Vor der Aktualisierung erledigt der Button allerdings noch eine andere, äußerlich nicht ersichtliches Aufgabe: Der Button liegt in einem anderen Formular als dem Formular, in dem der Filter eingestellt wurde. Eine Datenänderung im Formular «Filter» wird beim Verlassen des Formulars «Filter» und dem Betreten des Formular «MainForm» abgespeichert. Die Abspeicherung würde auch vorgenommen, wenn z.B. nur ein beliebiges Formularfeld des Formulars «MainForm» aufgesucht würde. Ein gesonderter Button zum Abspeichern des Wertes im Formular «Filter» ist also nicht erforderlich.

Das «Filter_Formular_Nullwerte» nutzt für das Listenfeld des Filters die Abfrage «Listenfeld_Kategorie_ohne_Eintrag». Dadurch können im Gegensatz zu dem vorhergehenden Formular auch nur die Datensätze gefiltert werden, die in der Tabelle "Medien" im Feld "katID" leer (**NULL**) sind.

Filter_Formular_Subformular_3Filter

Bereits das Formular «Filter_Formular_Subformular» filtert nach dem Verfasser im Subformular. Das untenstehende Formular birgt statt eines Listenfeldes für die Filterung gleich drei Listenfelder an.



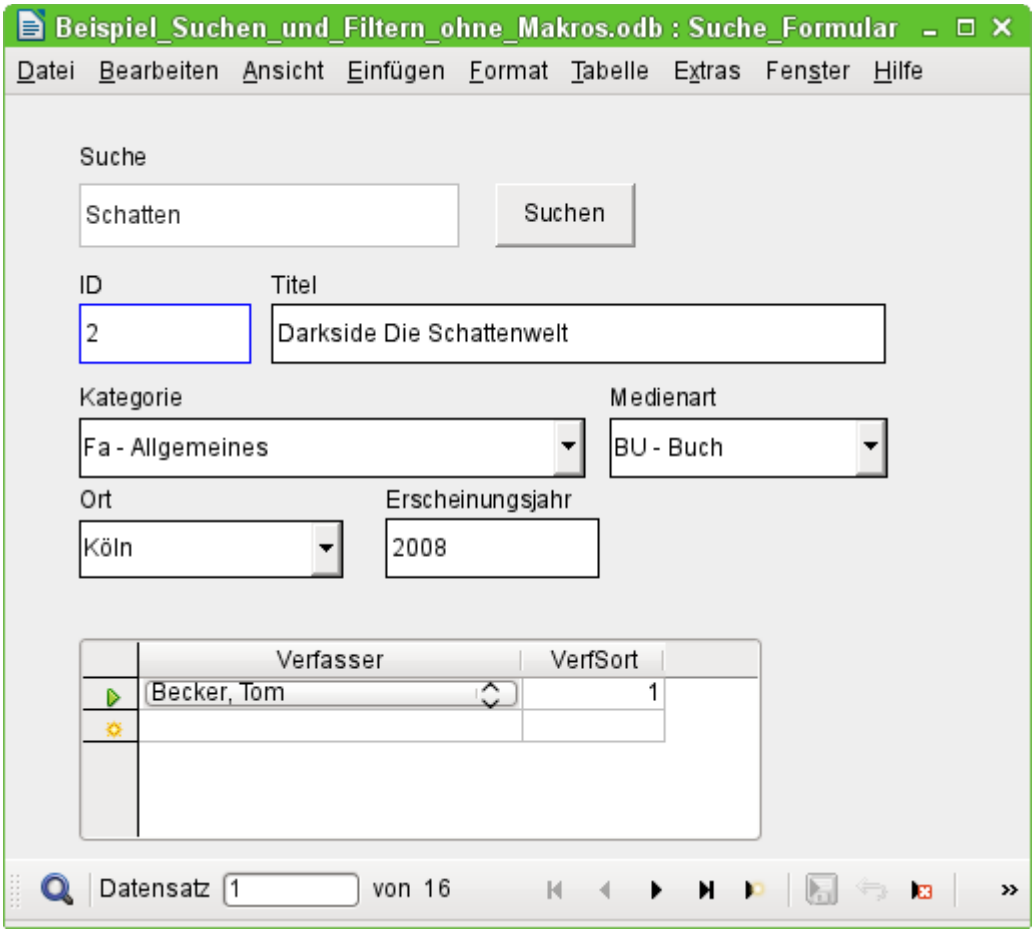
Die Listenfelder in diesem Formular sind nicht miteinander verbunden, d.h. der Eintrag des Wertes in einem Listenfeld sorgt nicht dafür, dass in dem anderen Listenfeld nur noch Werte angezeigt werden, die tatsächlich in der eingeschränkten Auswahl noch zur Verfügung stehen. Daher kann es bei direkter Betätigung aller drei Filter schnell dazu kommen, dass gar kein Datensatz mehr angezeigt wird, weil die Kombination einfach nicht vorhanden ist.

Es bietet sich für die Bedienung daher an, zuerst einmal nach einem Feld zu filtern und dann weiter zu sehen, welche Einträge überhaupt noch zur Auswahl stehen. Sich gegenseitig beeinflussende Listenfelder sind ohne den Einsatz von Makros nur mit vielen benutzerunfreundlichen Eingriffen in das Formular möglich. Dazu gehört vor allem eine laufende Betätigung des Buttons «Steuerelemente aktualisieren» in der Navigationsleiste für jedes verbleibende Listenfeld.

Datengrundlage für das Formular «MainForm» in diesem Formular «Filter_Formular_Subformular_3Filter» ist die oben vorgestellte Abfrage «Filter_Form_Subform_3Filter»

Suche_Formular

Äußerlich unterscheidet sich das «Suche_Formular» nur dadurch von dem «Filter_Formular», dass jetzt ein Texteingabefeld statt eines Listenfeldes zur Verfügung steht. Der Aufbau ist ansonsten identisch: Nebeneinander liegende Formulare «Filter» und «MainForm», Button zur Übernahme des Suchbegriffs als Aktualisierungsbutton in «MainForm» usw.

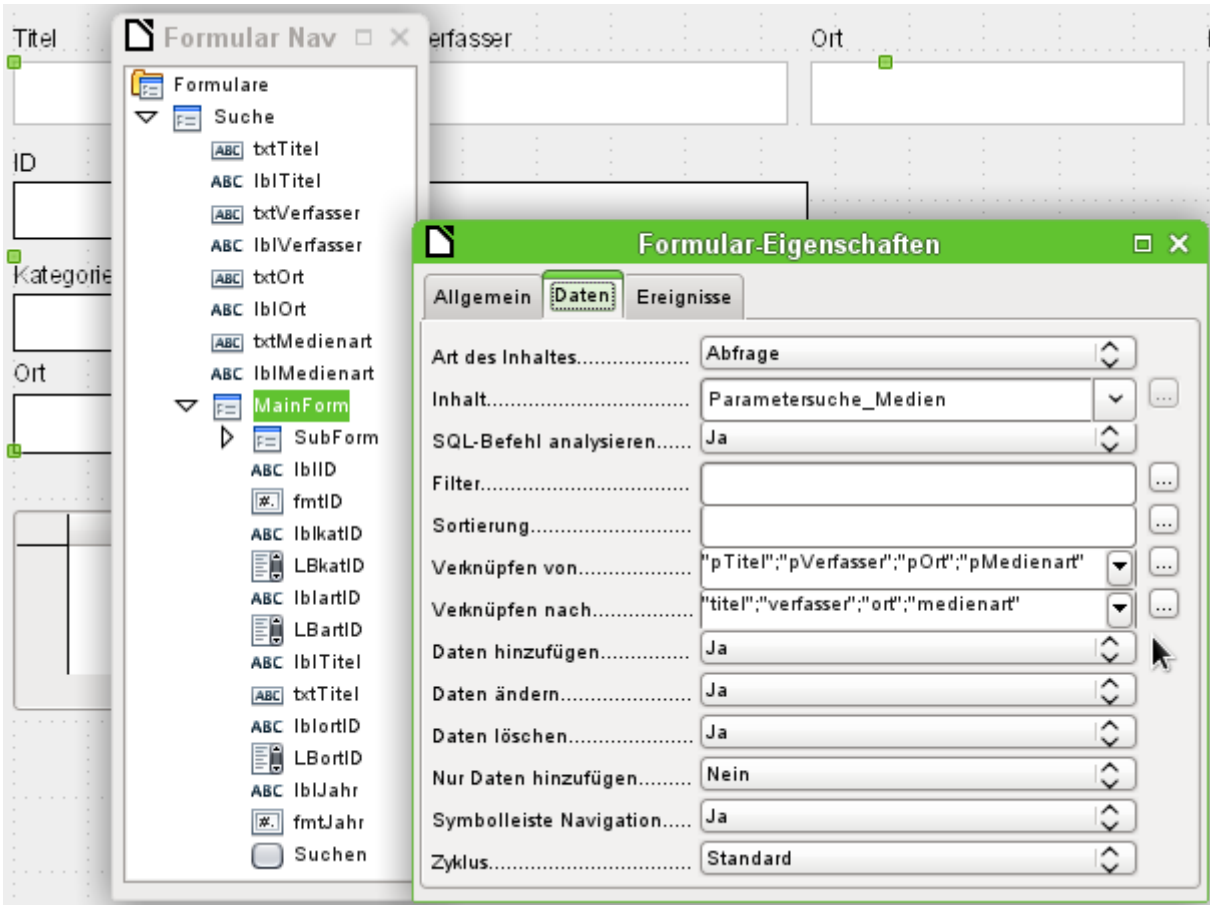


Das Nebenformular «MainForm» bezieht seine Daten aus der Abfrage «Suche_Form». Die Funktionsweise ist bei der Abfrage genauer erklärt.

In dem Formular «Suche_Formular_Subformular» ist lediglich die Datenquelle gegenüber dem gerade geschilderten Formular ausgetauscht. Die Datenquelle für dieses Formular ist die Abfrage «Suche_Formular_Subformular»

Parametersuche_Suche_Formular_Subformular

Die Parametersuche für ist gegenüber den vorhergehenden Suchformularen auf 4 Suchfelder erweitert worden. Äußerlich ist sie gleich aufgebaut. Der Unterschied ist erst sichtbar, wenn das Formular zum Bearbeiten geöffnet wird. Ein Blick auf die Ausgabe des Formular-Navigators zeigt, dass jetzt «MainForm» ein Unterformular des Formulars «Suche» ist. Es ist mit den Werten, die es aus dem Formular «Suche» bekommt, über die Felder für die Parametereingabe verbunden.



Die Felder "pTitel", "pVerfasser" usw. existieren in der Tabelle "Filter". Von ihnen aus wird nach den Parametern "titel", "verfasser" usw. verknüpft. Bei der Angabe der Parameter wird der führende Doppelpunkt (oder die ggf. gesetzte eckige Klammer als anderes Parametersymbol) nicht mit angegeben. Die Parameter werden also wie ein separates Feld der entsprechenden Tabelle gehandhabt.

Besondere Vorteile bietet die Weitergabe über Parameter gegenüber der direkten Arbeit mit der Filtertabelle aus dem Formular heraus nicht. Letztlich ist es dem Geschmack des Nutzers überlassen, zu welcher Art der Datensuche er/sie greifen möchte.

Suche_Filter_Formular_Subformular

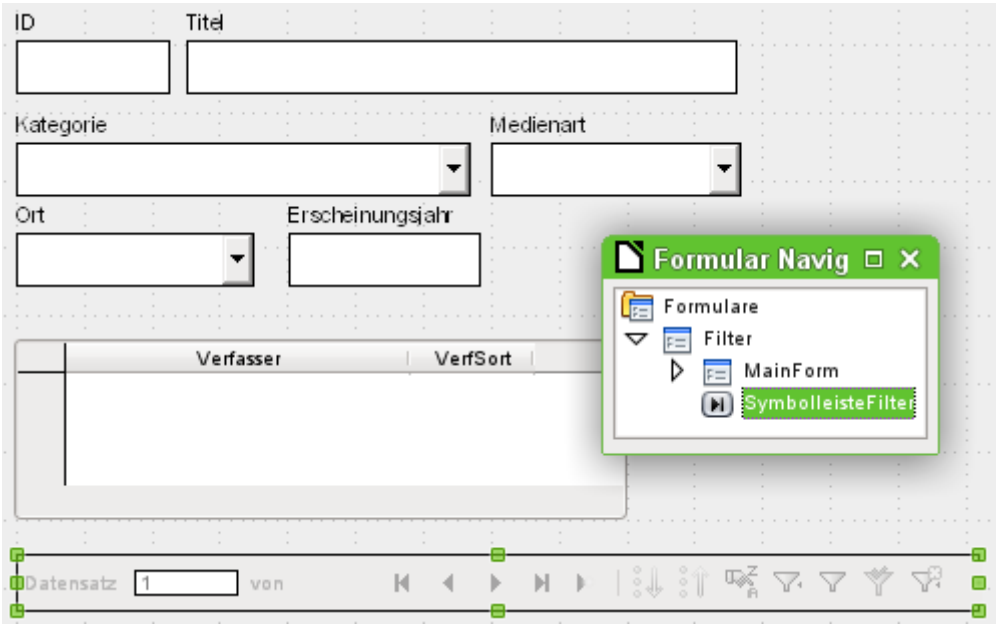
Das Formular «Suche_Filter_Formular_Subformular» ist schließlich eine Kombination aus Suchformular und Filterung mit drei Filtern.

Die Datenquelle für das Nebenformular «MainForm» ist die Abfrage «Suche_Filter_Form_Subform_3Filter»

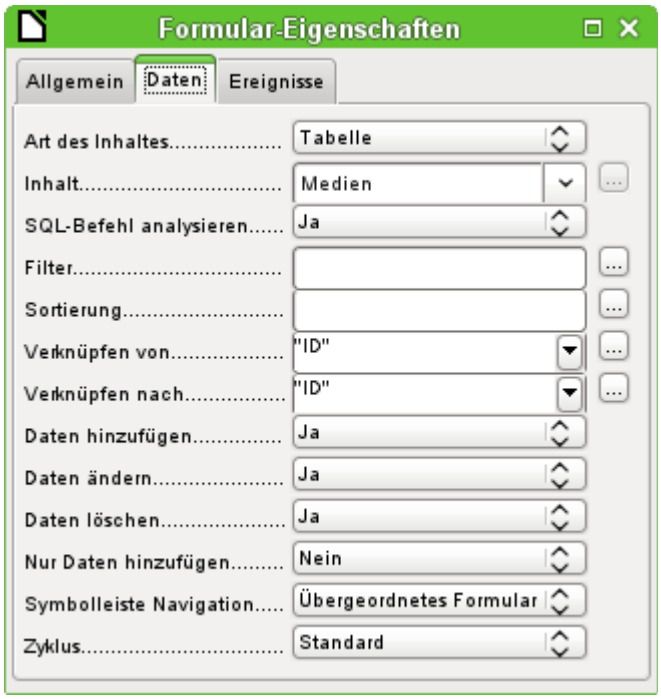
Standardfilter_Formular_Subformular

Die für ein Formular vorgesehenen Such- und Filtermöglichkeiten beschränken sich auf das Formular, in dem die jeweilige Filterfunktion aufgerufen wird. Inhalte eines Unterformulars lassen sich damit erst einmal nicht durchsuchen.

Dieses Formular geht deshalb einen anderen Weg. Die Datenbasis für das Hauptformular ist die "SuchAnsicht", wie sie auch in den anderen Formularen genutzt wird.

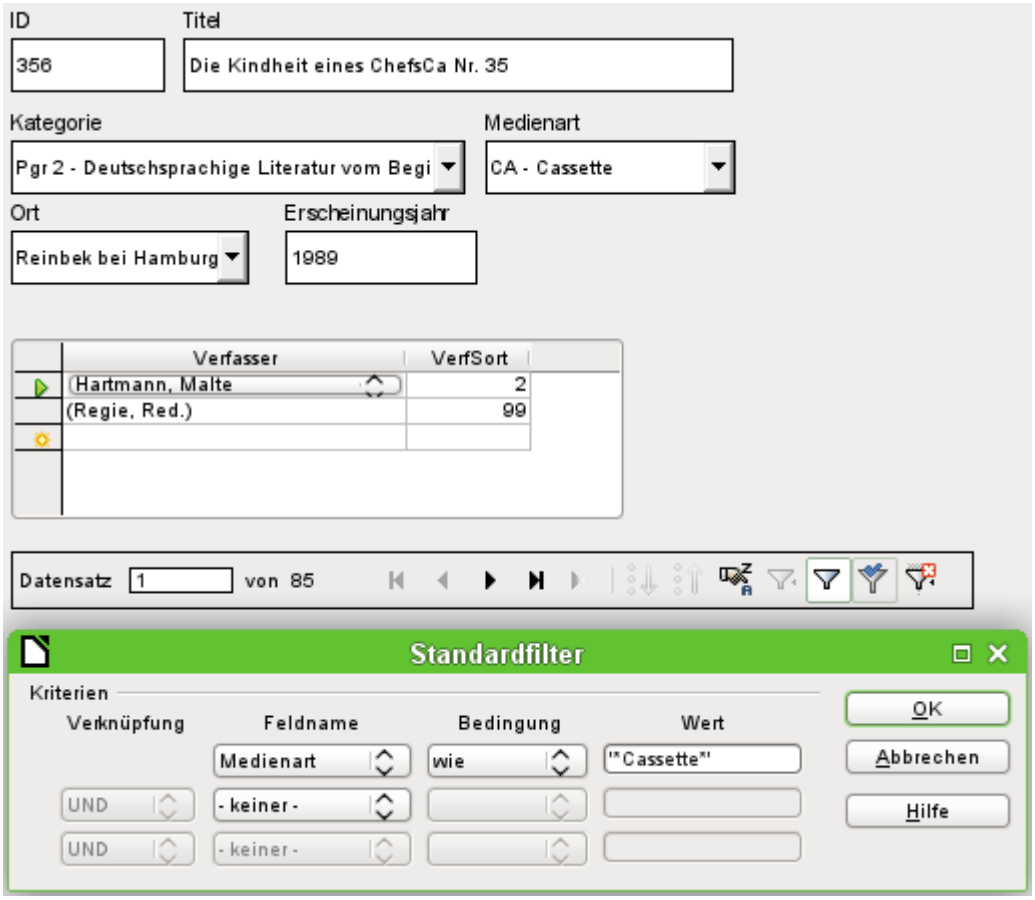


In dem Hauptformular «Filter» befindet sich nur eine Navigations-Symbolleiste. Über diese Symbolleiste kann die dem Hauptformular zugrundeliegende "SuchAnsicht" gefiltert werden.



Im Unterformular «MainForm» liegt die Tabelle "Medien". Das Unterformular ist über die "ID" mit dem Hauptformular verbunden. Da die "ID" das Primärschlüsselfeld der Tabelle "Medien" ist, wird in dem Unterformular grundsätzlich nur ein Datensatz angezeigt. Deshalb ist auch **Symbolleiste Navigation → Übergeordnetes Formular** gewählt. Dort erscheinen schließlich alle Felder, auf die die Filterung zutrifft.

Unterhalb von «MainForm» liegt, wie in den vorhergehenden Beispielen, «SubForm» zur Darstellung der Verfasser.



Hier wird nach der Medienart «Cassette» gefiltert. Der Standardfilter ist in seinen Funktionen im Handbuch erklärt. Bei den Werten muss leider jedes Mal ein entsprechender Text eingegeben werden. Einfacher wäre es, wenn nach Vorauswahl des Feldnamens und der Bedingung «=» ein Listenfeld erscheinen würde, das die möglichen Werte darstellt.

Für die Textsuche ist deshalb in der Regel die Kombination von **Bedingung → wie** mit **Wert → *Cassette*** am ehesten zielführend. Damit werden alle Datensätze angezeigt, die im Feld "Medienart" an irgendeiner Stelle das Wort «Cassette» stehen haben. Ist der genaue Inhalt allerdings bekannt, so kann entsprechend mit einem Gleichheitszeichen und einer genauen Eingabe gearbeitet werden.

Standardfilter_Formular_Subformular_Datensatzkorrektur

Äußerlich gleicht «Standardfilter_Formular_Subformular» diesem Formular. Beim Öffnen des Formular fällt lediglich auf, dass das Formular etwas langsamer startet. Dies liegt an der Datenquelle, die hier dem Hauptformular zugrunde liegt. Ein Blick auf die beiden Screenshots zeigt aber, dass im zweiten Formular statt vorher 85 Datensätzen nur noch 56 Datensätze bei gleichem Datenbe-

stand angezeigt werden. Tatsächlich enthält nämlich die Tabelle "Medien" nur 56 unterschiedliche Datensätze, die auf die Medienart «Cassette» verweisen. Alle weiteren Datensätze, die im vorhergehenden Formular aufgelistet werden, sind Duplikate. Diese Duplikate kommen dadurch zustande, dass in den Unterformularen mehrere Verfasser verzeichnet sind. Dies fällt schon bei dem dargestellten Datensatz in dem vorhergehenden Formular auf: Zwei Datensätze in dem Tabellenkontrollfeld für den Verfasser führen dazu, dass bei einer Navigation vom ersten zum zweiten Datensatz im Formular «Standardfilter_Formular_Subformular» kein Unterschied feststellbar ist.

Um den genauen Datensatzstand anzuzeigen und auch Irritationen bei der Datensatznavigation zu vermeiden muss deshalb der Inhalt der Felder "Verfasser" passend zu jedem Medium zu einem Feld zusammengefasst werden. Dazu ist weiter oben die Konstruktion mit Ansichten beschrieben.

Dem Hauptformular liegt nun die Ansicht "StandardfilterAnsicht" zugrunde, die gegenüber der "SuchAnsicht" schon auf die zu filternden Felder begrenzt ist und unnötige Felder gar nicht erst zur Filterung anbietet.

Diese "StandardfilterAnsicht" ist leider etwas langsamer im Aufbau, da der internen HSQLDB eine Funktion wie **`GROUP_CONCAT`**() (MySQL/MariaDB) oder **`LIST`** (Firebird) fehlt.

Suche und Filtern von Daten mit Makros erweitert

Bei der Erweiterung der Such- und Filterfunktionen mit Makros handelt es sich um die gleiche Datenbankkonstruktion wie beim Kapitel ohne Makroeinsatz. Die Tabellen sind in dem entsprechenden Kapitel beschrieben.

Abfragen

Die Abfragen müssen vor allem um Abfragen für die Listenfelder in Formularen erweitert werden. Listenfelder sollen so konstruiert sein, dass sie nur noch die Inhalte anzeigen, die bei der Filterung auch einen Treffer ergeben würden. Prinzipiell kann so etwas natürlich auch direkt in den Formularfeldern erfolgen. Die Abfragen wurden hier vorgezogen, da dort eine besser Einsicht in die Abfragekonstruktion besteht.

Listenfeld Medienart

Listenfeld_Medienart_bedingt

```
SELECT "Medienart", "ID"
FROM "Medienart"
WHERE "ID" IN ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Medienart" ASC
```

Zuerst wird das anzuzeigende Feld "Medienart" in der Abfrage dargestellt, dann das Primärschlüsselfeld, das später seinen Wert über das Formular an die Tabelle weitergeben soll.

Die angezeigten Werte in dem Listenfeld werden über die Unterabfrage auf die Werte beschränkt, die in der Abfrage "Filter_Form_Subform_3Filter" enthalten sind. Diese Abfrage wird, wie im vorhergehenden Kapitel beschrieben, durch die Filtereinstellungen beeinflusst und ist Datengrundlage für das Formular, in dem die gefilterten Daten dargestellt werden.

Bei jedem geänderten Wert in der Filtertabelle muss diese Abfrage über das Listenfeld neu eingelesen werden.

Listenfeld_Medienart_bedingt_Datensaetze

```
SELECT "Medienart" || ' → ' ||
      ( SELECT COUNT( "artID" )
      FROM "Filter_Form_Subform_3Filter"
      WHERE "artID" = "a"."ID"
```

```
      ) || ' Treffer' AS "Med",
   "ID"
FROM "Medienart" AS "a"
WHERE "ID" IN ( SELECT "artID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Med" ASC
```

Bei dieser Abfrage soll zusätzlich zu der Medienart im Listenfeld angegeben werden, wie viele Datensätze noch zur Auswahl stehen. Dazu ist hier eine korrelierende Unterabfrage eingebaut. Diese Abfrage ist über "a"."ID" mit dem Primärschlüssel zum jeweiligen dargestellten Datensatz verbunden.

Zuerst muss einmal der Tabelle **"Medienart"** ein Alias mit **AS "a"** zugewiesen werden. Dadurch kann mit einer Unterabfrage auf den jeweils aktuellen Datensatz zugegriffen werden. Die Unterabfrage zählt in der Abfrage "Filter_Form_Subform_3Filter" alle Datensätze, in der der Fremdschlüssel "artID" vorkommt, der gleich dem Primärschlüssel "ID" aus der Tabelle "Medienart" für den aktuellen Datensatz ist. Über Verkettungszeichen werden die verschiedenen darzustellenden Inhalte miteinander verbunden, damit sie als ein Feld in der Abfrage und später im Listenfeld erscheinen.

Listenfeld_Medienart_bedingt_Suche_Filter

```
SELECT "Medienart", "ID"
FROM "Medienart"
WHERE "ID" IN
   ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
ORDER BY "Medienart" ASC
```

Der einzige Unterschied zur Abfrage «Listenfeld_Medienart_bedingt» ist bei dieser Abfrage lediglich die Datenquelle, auf die sich die Unterabfrage bezieht. Statt "Filter_Form_Subform_3Filter" ist die Datenquelle hier "Suche_Filter_Form_Subform_3Filter", die eben neben den Filtern auch noch ein Suchfeld abdeckt.

Listenfeld_Medienart_bedingt_Suche_Filter_Datensaetze

```
SELECT "Medienart" || ' → ' ||
      ( SELECT COUNT( "artID" )
      FROM "Suche_Filter_Form_Subform_3Filter"
      WHERE "artID" = "a"."ID"
      ) || ' Treffer' AS "Med",
   "ID"
FROM "Medienart" AS "a"
WHERE "ID" IN
   ( SELECT "artID" FROM "Suche_Filter_Form_Subform_3Filter" )
ORDER BY "Med" ASC
```

Die Abfrage entspricht der Abfrage «Listenfeld_Medienart_bedingt_Datensaetze». Statt der Abfrage «Filter_Form_Subform_3Filter» dient hier aber die Abfrage «Suche_Filter_Form_Subform_3Filter» als Datengrundlage. Damit ist neben der Filterung auch die Suchfunktion enthalten.

Listenfeld Kategorie

Für das Listenfeld «Kategorie» sind vier unterschiedliche Abfragen erstellt worden. Die Abfrage, die auch die Felder ohne Eintrag auflistet, ist bereits im Kapitel ohne Makroeinsatz erklärt. Hier die weiteren Abfragen:

Listenfeld_Kategorie_bedingt

```
SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"
FROM "Kategorien"
WHERE "ID" IN ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Kat" ASC
```

Der Unterschied zwischen «Listenfeld_Medienart_bedingt» und «Listenfeld_Kategorie_bedingt» besteht neben den unterschiedlichen Datenquellen in der Darstellung lediglich darin, dass sowohl die systematische Einordnung "System" als auch die dazugehörige Beschreibung "Kategorie" gemeinsam in dem Listenfeld sichtbar erscheinen.

Die Auswahl der angezeigten Kategorien wird durch die Abfrage "Filter_Form_Subform_3Filter" eingeschränkt, so dass nur Kategorien angezeigt werden, die auch in der Tabelle "Medien" tatsächlich vorhanden sind.

Listenfeld_Kategorie_bedingt_Datensaetze

```
SELECT "System" || ' - ' || "Kategorie" || ' → ' ||
   ( SELECT COUNT( "katID" ) FROM "Filter_Form_Subform_3Filter"
      WHERE "katID" = "a"."ID"
   ) || ' Treffer' AS "Kat",
   "ID"
FROM "Kategorien" AS "a"
WHERE "ID" IN
   ( SELECT "katID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Kat" ASC
```

Der Unterschied zwischen «Listenfeld_Medienart_bedingt_Datensaetze» und «Listenfeld_Kategorie_bedingt_Datensaetze» besteht neben den unterschiedlichen Datenquellen lediglich darin, dass sowohl die systematische Einordnung "System" als auch die dazugehörige Beschreibung "Kategorie" gemeinsam in dem Listenfeld sichtbar erscheinen.

Listenfeld_Kategorie_bedingt_Suche_Filter

```
SELECT "System" || ' - ' || "Kategorie" AS "Kat", "ID"
FROM "Kategorien"
WHERE "ID" IN
   ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )
ORDER BY "Kat" ASC
```

Der einzige Unterschied zur Abfrage «Listenfeld_Kategorie_bedingt» ist bei dieser Abfrage lediglich die Datenquelle, auf die sich die Unterabfrage bezieht. Statt "Filter_Form_Subform_3Filter" ist die Datenquelle hier "Suche_Filter_Form_Subform_3Filter", die eben neben den Filtern auch noch ein Suchfeld abdeckt.

Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze

```
SELECT "System" || ' - ' || "Kategorie" || ' → ' ||
   ( SELECT COUNT( "katID" ) FROM "Suche_Filter_Form_Subform_3Filter"
      WHERE "katID" = "a"."ID"
   ) || ' Treffer' AS "Kat",
   "ID"
FROM "Kategorien" AS "a"
WHERE "ID" IN
   ( SELECT "katID" FROM "Suche_Filter_Form_Subform_3Filter" )
ORDER BY "Kat" ASC
```

Die Abfrage entspricht der Abfrage «Listenfeld_Kategorie_bedingt_Datensaetze». Statt der Abfrage «Filter_Form_Subform_3Filter» dient hier aber die Abfrage «Suche_Filter_Form_Subform_3Filter» als Datengrundlage. Damit ist neben der Filterung auch die Suchfunktion enthalten.

Listenfeld Verfasser

Listenfeld_Verfasser_bedingt

```
SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"
FROM "rel_Medien_Verfasser", "Verfasser"
```

```
WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
   AND "rel_Medien_Verfasser"."VerfSort" < 99
   AND "rel_Medien_Verfasser"."medID" IN
      ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Verfasser"."Verfasser" ASC
```

Bei der Abfrage der Verfasser sollen die Einträge ausgeschlossen werden, bei denen in der Tabelle "rel_Medien_Verfasser" der Eintrag **`"VerfSort" >= 99`** ist. Es handelt sich dabei um Zusatzinformationen wie z.B. «(Hrsg.)».

Um diese Einträge auszufiltern bezieht diese Abfrage ihre Informationen aus zwei Tabellen. Die Tabellennamen müssen also auf jeden Fall vor die Feldbezeichnungen geschrieben werden. Außerdem muss mit **`DISTINCT`** ausgeschlossen werden, dass später Doppler in dem Listenfeld erscheinen.

Listenfeld_Verfasser_bedingt_Datensaetze_langsam

```
SELECT DISTINCT "Verfasser" || ' → ' ||
   ( SELECT COUNT( "vefID" )
      FROM "SuchAnsicht"
      WHERE "vefID" = "a"."vefID" AND "ID" IN
         ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
      ) || ' Treffer' AS "Ver",
   "vefID"
FROM "SuchAnsicht" AS "a"
WHERE "VerfSort" < 99
   AND "ID" IN ( SELECT "ID" FROM "Filter_Form_Subform_3Filter" )
ORDER BY "Ver" ASC
```

Der entscheidende Nachteil beim Auszählen der Treffer besteht darin, dass die Tabelle "Verfasser" nicht direkt mit der Tabelle "Medien" verbunden ist. Dies liegt daran, dass ein Medium auch mehrere Verfasser haben kann, das Ganze also eine n:m-Beziehung ist.

Die obige Abfrage stellt einen von vielen Versuchen dar, neben dem Verfasser aufzuzeigen, wie viele der verbleibenden Datensätze Inhalte haben, die zu dem Verfasser stammen. Dazu wurde bereits, statt direkt auf die Tabellen "Verfasser" und "rel_Medien_Verfasser" Bezug zu nehmen, die Auflistung der Verfasser in der "SuchAnsicht" zu Hilfe genommen.

Durch die Abfrage der Anzahl, die aus einer Unterabfrage und eine darin liegenden weiteren Unterabfrage besteht, wurde die gesamte Abfrage viel zu langsam. Dies zeigte sich besonders bei entsprechend vielen Datensätzen. Entscheidend für diese Bremse ist die Unterabfrage innerhalb der Unterabfrage, die keinen eindeutigen Treffer in Form eines einzelnen Datensatzes ergibt. **`SELECET "ID" FROM "Filter_Form_Subform_3Filter"`** zeigt jedes Mal ca. 5000 Datensätze auf, in denen nach einer Übereinstimmung gesucht wird. Bei fast 2000 Datensätzen aus "Verfasser" ergibt das für eine leere Filter-Tabelle 2000*5000 = 10000000 Abfragen.

Aus diesem Grund lässt sich mit der obigen Abfrage die Anzahl der Datensätze nicht in vertretbarer Zeit in einem Listenfeld darstellen. Das Öffnen eines Formulars mit so einem Listenfeld würde dazu führen, dass davon ausgegangen wird, dass LO abgestürzt ist.

Listenfeld_Verfasser_bedingt_Datensaetze_optimiert

```
SELECT DISTINCT "Verfasser" || ' → ' ||
   ( SELECT COUNT( "vefID" )
      FROM "Filter_Ansicht"
      WHERE "vefID" = "a"."vefID"
   ) || ' Treffer' AS "Ver",
   "vefID"
FROM "Filter_Ansicht" AS "a"
```

```
WHERE "VerfSort" < 99
ORDER BY "Ver" ASC
```

Hier wurde statt der der Verschachtelung von 2 Unterabfragen nur eine einfache Unterabfrage genutzt. Während in der Abfrage "Filter_Form_Subform_3Filter" kein Feld für den Fremdschlüssel des Verfassers "vefID" auftauchte, ist hier eine gesonderte Abfrage "Filter_Ansicht" erstellt worden, die dieses zusätzliche Feld bereit stellt. Diese gesonderte Abfrage ist weiter unten erklärt. Sie ist nicht für die weitere Nutzung im Formular geeignet, weil sie deutlich mehr Datensätze darstellt, als in der Tabelle "Medien" verzeichnet sind – eben weil es mehrere Verfasser für ein Medium geben kann.

Durch die Beziehung auf nur eine Unterabfrage wird die gesamte Abfrage um ein vielfaches schneller, bremst aber leider später im Formular immer noch etwas den Aufbau des Formulars. Ein Zeichen dafür, dass sich eine Trefferauflistung wohl nicht so gut für ein großes Datenvorkommen in einem Listenfeld eignet.

Vom Prinzip her ist die obige Abfrage jetzt gleich der Abfrage «Listenfeld_Medienart_bedingt_Datensaetze». Es ist lediglich zusätzlich die Einschränkung bei den darzustellenden Bezeichnungen für die "Verfasser" vorgesehen: "VerfSort" < 99. Da die Abfrage «Filter_Such_Ansicht» bereits die Filterung berücksichtigt muss nicht mehr ermittelt werden, ob der angezeigte Verfasser in einer bestimmten Datenmenge enthalten ist.

Listenfeld_Verfasser_bedingt_Suche_Filter

```
SELECT DISTINCT "Verfasser"."Verfasser", "Verfasser"."ID"
FROM "rel_Medien_Verfasser", "Verfasser"
WHERE "rel_Medien_Verfasser"."vefID" = "Verfasser"."ID"
   AND "rel_Medien_Verfasser"."VerfSort" < 99
   AND "rel_Medien_Verfasser"."medID" IN
      ( SELECT "ID" FROM "Suche_Filter_Form_Subform_3Filter" )
ORDER BY "Verfasser"."Verfasser" ASC
```

Der einzige Unterschied zur Abfrage «Listenfeld_Verfasser_bedingt» ist bei dieser Abfrage lediglich die Datenquelle, auf die sich die Unterabfrage bezieht. Statt "Filter_Form_Subform_3Filter" ist die Datenquelle hier "Suche_Filter_Form_Subform_3Filter", die eben neben den Filtern auch noch ein Suchfeld abdeckt.

Listenfeld_Verfasser_bedingt_Suche_Filter_Datensaetze

```
SELECT DISTINCT "Verfasser" || ' → ' ||
   ( SELECT COUNT( "vefID" )
      FROM "Filter_Such_Ansicht"
      WHERE "vefID" = "a"."vefID"
   ) || ' Treffer' AS "Ver",
   "vefID"
FROM "Filter_Such_Ansicht" AS "a"
WHERE "VerfSort" < 99
ORDER BY "Ver" ASC
```

Die Abfrage entspricht der Abfrage «Listenfeld_Verfasser_bedingt_Datensaetze_optimiert». Statt der Abfrage «Filter_Ansicht» dient hier aber die Abfrage «Filter_Such_Ansicht» als Datengrundlage. Damit ist neben der Filterung auch die Suchfunktion enthalten.

Filter_Ansicht

```
SELECT *
FROM "SuchAnsicht"
WHERE "ID" IN (
   SELECT DISTINCT "ID"
   FROM "SuchAnsicht"
```

```
   WHERE "VerfasserID" = IFNULL(
      ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
      "VerfasserID" )
   AND "MedienartID" = IFNULL(
      ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" = TRUE ),
      "MedienartID" )
   AND "KategorieID" = IFNULL(
      ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
      "KategorieID" )
   )
```

Diese Abfrage nutzt statt der Tabelle "Medien" die Ansicht "SuchAnsicht" als Datenquelle. In der "SuchAnsicht" sind alle Informationen der Tabellen zusammen gefasst. Dadurch erscheinen auch die "Verfasser" sowie die Fremdschlüssel für die "Verfasser" zur Auswahl. Das wird dann in der Abfrage «Listenfeld_Verfasser_bedingt_Datensaetze_optimiert» genutzt, um eine Unterabfrage innerhalb einer Unterabfrage erstellen zu können.

«Filter_Ansicht» filtert die Datensätze bereits vor, so dass die Listenfeldabfrage noch zusätzlich um eine Unterabfrage verkürzt werden kann.

«Filter_Ansicht» könnte natürlich auch für die anderen Listenfelder verwendet werden. Die Notwendigkeit dieser Abfrage hat sich allerdings erst ergeben, als eben die Abfrage der Verfasser zu träge wurde. Diese Datenbank soll schließlich aufzeigen, welche Möglichkeiten es gibt und was dabei zu berücksichtigen ist. Während für alle Felder, die bereits indirekt in der Tabelle "Medien" über Fremdschlüssel vorhanden waren, der Bezug auf eine Abfrage, die nur diese Tabelle darstellt, ausreicht, muss eben für dort nicht vorhandene Felder ein zusätzlicher Weg beschritten werden.

Filter_Such_Ansicht

```
SELECT *
FROM "SuchAnsicht"
WHERE "ID" IN (
   SELECT DISTINCT "ID"
   FROM "SuchAnsicht"
   WHERE "VerfasserID" = IFNULL(
      ( SELECT "Filter_Verfasser" FROM "Filter" WHERE "ID" = TRUE ),
      "VerfasserID" )
   AND "MedienartID" = IFNULL(
      ( SELECT "Filter_Medienart" FROM "Filter" WHERE "ID" = TRUE ),
      "MedienartID" )
   AND "KategorieID" = IFNULL(
      ( SELECT "Filter_Kategorie" FROM "Filter" WHERE "ID" = TRUE ),
      "KategorieID" )
   AND (
      LOWER ( "Titel" ) LIKE IFNULL( '%' || LOWER (
         ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%',
         LOWER ( "Titel" ) )
      OR LOWER ( "Kategorie" ) LIKE '%' || LOWER (
         ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Medienart" ) LIKE '%' || LOWER (
         ( SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Ort" ) LIKE '%' || LOWER ( (
         SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
      OR LOWER ( "Verfasser" ) LIKE '%' || LOWER ( (
         SELECT "Suche" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
      )
   )
```

Diese Abfrage startet genau wie die Abfrage «Filter_Ansicht». Datenquelle ist die "SuchAnsicht". Aus der "SuchAnsicht" werden alle Felder ausgelesen, bei denen die Filterwerte aus "VerfasserID", "MedienartID" und "KategorieID" mit den Einträgen in der Tabelle "Filter" übereinstimmen, sofern in der Tabelle "Filter" für diese Felder ein Eintrag existiert. Die Existenz des Eintrages wird mit **IFNULL** überprüft.

An den Inhalt aus «Filter_Ansicht» wird schließlich noch die Suchbedingung angefügt. Sofern das Feld "Filter"."Suche" einen Eintrag vorweist werden die Felder "Titel", "Kategorie", "Medienart", "Ort" und "Verfasser" daraufhin untersucht, ob der eingegebene Text in einem ihrer Felder unabhängig von der Position und von Groß- und Kleinschreibung enthalten ist. Durch die **OR**-Verknüpfung der Felder "Titel", "Kategorie", "Medienart", "Ort" und "Verfasser" muss der Suchbegriff nur in einem Feld vertreten sein, um das Medium in der Abfrage zurück zu geben.

Direkte Filterung

Die direkte Filterung eines Formulars kommt ohne die Abspeicherung von Daten aus. Hierfür sind zwei Abfragen vorgesehen.

Filter_Form_Nullwerte

```
SELECT "Medien".*, IFNULL( "katID", 0 ) AS "kat" FROM "Medien"
```

Alle Datensätze aus der Tabelle "Medien" werden aufgelistet. Daneben wird noch ein Feld mit der Bezeichnung "kat" geschrieben, dass die Werte von "katID" übernimmt. Ist "katID" allerdings leer, so wird dort eine 0 eingetragen. Damit kann durch eine Suche nach dem Wert 0 über ein Listenfeld auch nach leeren Feldern gefiltert werden.

Diese Abfrage entspricht dem Beginn der Abfrage «Filter_Form». Bei der Abfrage «Filter_Form» ist allerdings noch eine Bedingung genannt, nach der entsprechende Werte über einen Eintrag in der Tabelle "Filter" gefiltert werden.

Filter_Leerfeld

```
SELECT "Filter".*, ( SELECT NULL FROM "Filter" ) AS "Leer"
FROM "Filter" WHERE "ID" = TRUE
```

Die Tabelle "Filter" hat für eine direkte Filterung den Nachteil, dass sie einmal ausgewählte Werte fest speichert. Dadurch erscheinen beim Öffnen eines Formulars Filtereinstellungen, nach denen sich der direkte Filter aber nicht richtet. Schließlich basiert das entsprechende Formular auf einer Abfrage, die gar nicht erst den Eintrag in der Filtertabelle nutzt.

Am einfachsten wäre es, ein Formularfeld ohne Datengrundlage für die Filterung zu nutzen. Das wäre zu Beginn dann immer leer. Solche Felder lassen sich aber leider nicht erstellen. Daher wird hier über eine Unterabfrage ein leeres Feld (**NULL**) mit der Bezeichnung "Leer" erstellt. Base behandelt dies erst einmal wie ein beschreibbares Datenfeld. Die Verbindung eines Listenfeldes im Formular zu diesem Datenfeld ist möglich. Das Listenfeld ist dadurch bei Formularstart immer leer. Ohne die Unterabfrage versucht Base im Formular dennoch eine Speicherung vorzunehmen und erkennt dann, dass gar kein Wert geändert wurde.

Da eine Abfrage sich immer auf eine Tabelle beziehen muss wurde hier die Tabelle "Filter" als «Datenquelle» genutzt. Sie hat schließlich nur einen Datensatz und ist damit am schnellsten verfügbar. Da die Abfrage auch den Primärschlüssel der Tabelle "Filter" enthält ist das Feld "Leer" für die grafische Benutzeroberfläche scheinbar editierbar.

Formulare

Erstes Kennzeichen sämtlicher Formulare in dieser Datenbank ist im Unterschied zu den vorhergehenden gleichlautenden Formularen aus der Datenbank «Beispiel_Suchen_und_Filtern_ohne_Makros» das Fehlen eines Buttons, der die Filterung bzw. Suche startet.

Das Formular «Filter_Formular»

Wird der Wert im Listenfeld auf einen anderen Wert eingestellt, so wird automatisch das Formular auf diesen Filterwert eingestellt. Das Makro dafür ist an das Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.

Markosteuerung für das automatische Filtern

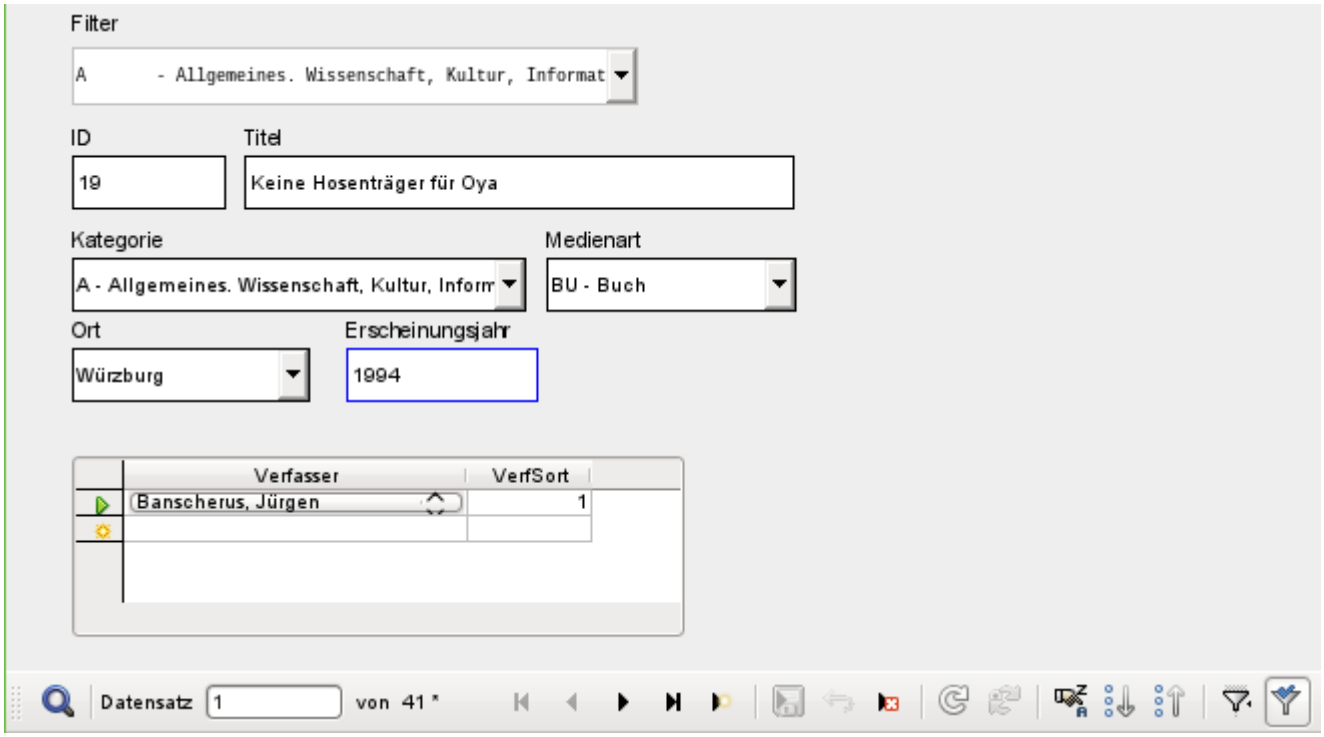
```
SUB Filter
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeldList AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm1 = oDrawpage.forms.getByName("Filter")
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeldList = oForm1.getByName("lboFilter")
    oFeldList.commit()
    oForm1.updateRow()
    oForm2.reload()
END SUB
```

Die Zugriffe zu den beiden nebeneinander stehenden Formularen werden definiert. Das Listenfeld für die Filterung mit dem Namen «lboFilter» liegt im Formular «Filter». Der eingestellte Wert des Listenfeldes wird in dem aktuellen Datensatz des Formulars durch **`commit`**() abgelegt. Er muss jetzt noch in den Datensatz geschrieben werden. Dies geschieht durch **`updateRow`**() im entsprechenden Formular. Es handelt sich hier um ein Update, da ja keine neuen Datensätze geschrieben werden, sondern nur der (einzige) Datensatz mit dem Primärschlüssel **`"ID" = TRUE`** geändert wird.

Nach dem Abspeichern wird das Formular, das das Ergebnis der Filterung anzeigen soll, mit **`reload`**() neu eingelesen.

Das Formular «Filter_Formular_direkt_Nullwerte»

Auch hier wird durch Änderung des Wertes im Listenfeld automatisch das Formular auf diesen Filterwert eingestellt. Das Makro dafür ist an das Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.



Bei genauerem Hinsehen auf das Formular wird allerdings deutlich, dass in der Navigationsleiste jetzt eine aktive Filterung markiert ist (rechts unten in). Der direkt eingegebene Filter kann hier ein- und wieder ausgeschaltet werden. Er kann auch über den entsprechenden Filterbutton gelöscht werden.

Markosteuerung für die direkte Filterung des Formulars

```
SUB Filter_Formular_direkt
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeldList AS OBJECT
    DIM stListValue AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm1 = oDrawpage.forms.getByName("Filter")
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeldList = oForm1.getByName("lboFilter")
    stListValue = oFeldList.getCurrentValue()
    IF stListValue = "" THEN
        oForm2.Filter = ""
    ELSE
        oForm2.Filter = "kat = '" + stListValue + "'"
    END IF
    oForm2.reload()
END SUB
```

Auch hier wird wieder zuerst der Zugriff auf die Formulare und das Listenfeld erstellt. Anschließend wird der Wert des Listenfeldes ausgelesen. Dies ist nicht der angezeigte Text, sondern (seit LO 4.1) der Inhalt, der an die dem Formular zugrundeliegende Tabelle weiter gegeben werden soll. In diesem Falle handelt es sich also um den Primärschlüssel der Tabelle "Kategorie".

Wird im Listenfeld das leere Feld gewählt (Listenfeld ganz oben), so ist auch der weitergegebene Inhalt leer. Da die Variable als Text deklariert wurde reicht hier ein **`stListValue = ""`**, um diesen Zustand zu erkennen. Bei einem leeren Filterwert soll der Filter ausgeschaltet werden: **`oForm2.Filter = ""`**.

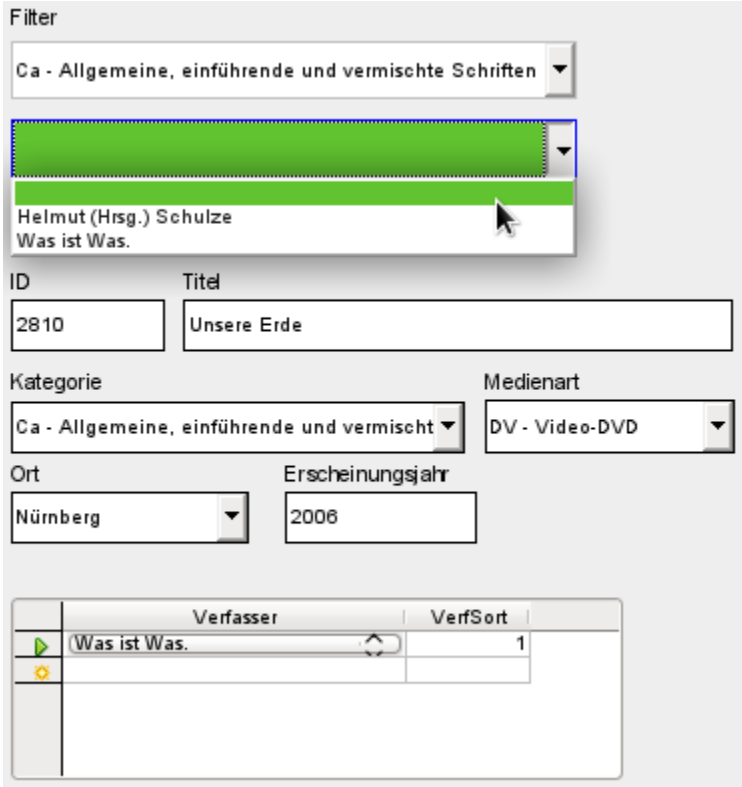
Wird im Listenfeld ein Eintrag gewählt, so wird der entsprechende Schlüssel an die Filterformulierung weiter gegeben. Der gesamte Filter lautet dann z.B. **`"kat = '13'"`**. Nach Einstellung des Filters wird das Hauptformular wieder neu geladen. Damit wird der Filter übernommen.

Das Formular «Filter_Formular_Subformular»

Lediglich die Datenquelle des Hauptformulars unterscheidet sich in diesem Formular von dem Formular «Filter_Formular». Das Makro sowie die Einbindung des Makros in das Formular sind identisch.

Das Formular «Filter_Formular_Subformular_bedingende_Filter»

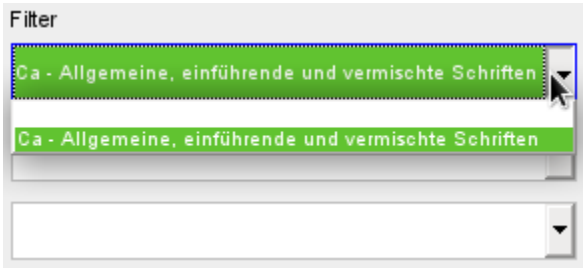
Dieses Formular hat 3 Listenfelder, bei denen die Filter entsprechend ausgesucht werden können.



Durch die Filterung im ersten Listenfeld sind im zweiten und im dritten Listenfeld nur noch die Werte verfügbar, die auch noch in den verbleibenden Datensätzen des Formulars vorhanden sind. In der Kategorie 'Ca – Allgemeine, einführende und vermischte Schriften' sind also nur von zwei verschiedenen Autoren Werke verzeichnet.

Datengrundlage dieser Listenfelder sind die Abfragen «Listenfeld_Kategorie_bedingt», «Listenfeld_Verfasser_bedingt» und «Listenfeld_Medienart_bedingt».

Das Makro «Filter_bedingt» ist an jedes Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.



Dieses Makro hat bei genauer Hinsicht aber noch einen Nachteil, der in den folgenden Formularen beseitigt werden muss: Sobald bei einem Listenfeld eine Auswahl getroffen wurde (hier bei der Kategorie), wird das entsprechende Listenfeld ebenso wie die anderen aktualisiert. Dadurch wird dort nur noch der gerade ausgewählte Filterwert angezeigt. Nur wenn das Feld wieder als leeres Feld eingestellt wird erscheinen auch die entsprechenden anderen Auswahlmöglichkeiten erneut zur Auswahl.

Markosteuerung für die bedingte Filterung des Formulars

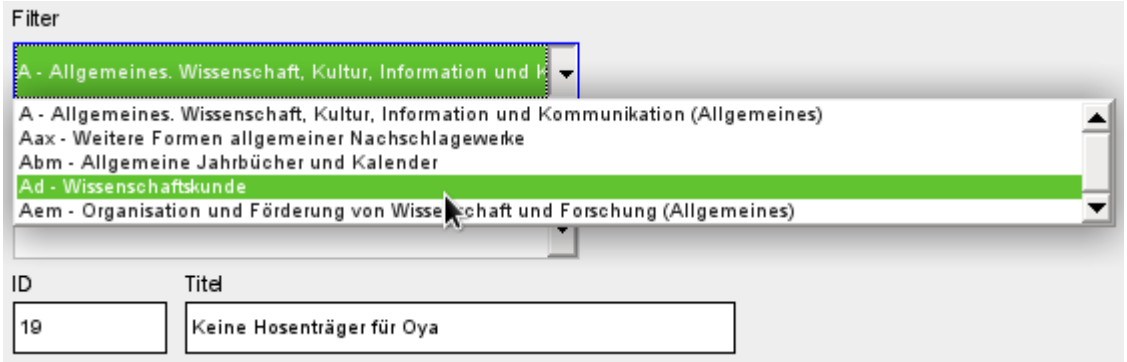
Dieses Makro ersetzt den Aktualisierungsbutton in dem zu filternden Formular. Die Einstellung eines Filters beeinflusst gleichzeitig die anzuzeigenden Inhalte der anderen Filter. Dadurch wird die Auswahl der anderen Filter auf die möglichen Treffer reduziert.

```
SUB Filter_bedingt
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeldList1 AS OBJECT
    DIM oFeldList2 AS OBJECT
    DIM oFeldList3 AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm1 = oDrawpage.forms.getByName("Filter")
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeldList1 = oForm1.getByName("lboFilter1")
    oFeldList2 = oForm1.getByName("lboFilter2")
    oFeldList3 = oForm1.getByName("lboFilter3")
    oFeldList1.commit()
    oFeldList2.commit()
    oFeldList3.commit()
    oForm1.updateRow()
    oFeldList1.refresh()
    oFeldList2.refresh()
    oFeldList3.refresh()
    oForm2.reload()
END SUB
```

Die Variablen werden deklariert und der Zugang zum Filter-Formular, jedem einzelnen Listenfeld sowie dem Formular, das das Filterergebnis darstellen soll erstellt. Unabhängig davon, welcher Filter ausgelöst wurde, werden von allen Filtern die Werte in die dem Formular zugrundeliegende Filtertabelle übertragen: **`commit()`**. Anschließend wird der geänderte Datensatz abgespeichert: **`updateRow()`**. Danach können die Listenfeld neu eingelesen werden, da ihr Inhalt auf die Einträge in der Filtertabelle zugreift: **`refresh()`**. Schließlich muss nur noch das Formular zur Darstellung des Filterergebnisses mit **`reload()`** neu geladen werden.

Das Formular «Filter_Formular_Subformular_bedingende_Filter_allround»

Äußerlich gleich das Formular dem vorhergehenden. Wird allerdings ein Wert in einem Listenfeld ausgesucht, so wird dieses Listenfeld nicht direkt aktualisiert. Dadurch ergibt sich im Unterschied zum vorhergehenden Jahr folgendes Bild:



Ausgesucht war die Kategorie 'A – Allgemeines …'. Das zeigt die aktuelle Anzeige im Listenfeld. Gleichzeitig ist das Listenfeld aber weiterhin mit allen anderen Werten gefüllt, so dass ohne einen Zwischenstopp über ein leeres Listenfeld direkt ein anderer Wert ausgesucht werden kann.

Der Listeninhalt entspricht ebenfalls dem vom vorhergehenden Formular «Filter_Formular_Subformular_bedingende_Filter».

In den Eigenschaften der Listenfelder ist allerdings ein kleiner Unterschied zu verzeichnen:

In den Zusatzinformationen ist der Name der Listenfelder abgespeichert, die aktualisiert werden sollen. Die Bezeichnungen sind durch ein Komma getrennt. Hier, beim Listenfeld lboFilter1, steht also **`lboFilter2,lboFilter3`**. Diese Information wird im Makro ausgewertet

Das Makro «Filter_bedingt_allround» ist an jedes Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.

Markosteuerung für die bedingte Allround-Filterung des Formulars

Mit diesem Makro werden die anderen Listenfelder vom Inhalt her neu eingelesen, wenn ein Listenfeld von der Auswahl her geändert wurde. Das Listenfeld, dessen Auswahl geändert wurde, wird nur dann neu eingelesen, wenn das leere Feld ausgewählt wurde, also nur eine Auswahl komplett zurück genommen wurde.

```
SUB Filter_bedingt_allround(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM stTag AS String
    DIM arList()
    DIM i AS INTEGER
```

```
    oFeld = oEvent.Source.Model
    oForm1 = oFeld.Parent
    stTag = oFeld.Tag
    arList = Split(stTag, ",")
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeld.commit()
    oForm1.updateRow()
    FOR i = LBound(arList()) TO UBound(arList())
       oForm1.getByName(arList(i)).refresh()
    NEXT
    IF IsEmpty(oFeld.currentValue) THEN
       oFeld.refresh()
    END IF
    oForm2.reload()
END SUB
```

Bereits der Aufruf des Makros berücksichtigt, von welchem Listenfeld aus das Signal kam: **`oEvent AS OBJECT`**. Anschließend werden die Variablen deklariert. Besonders hinzuweisen ist hier auf die Deklaration des Arrays, die im voraus erfolgen muss: **`arList()`**.

Das Feld wird über das auslösende Ereignis ermittelt. Das entsprechende Formular ist direkt dem Feld übergeordnet: **`oFeld.Parent`**. Entsprechend muss auch der Name des Formulars nicht mehr gesondert im Makro erscheinen.

Aus den Zusatzinformationen sollen die Bezeichnungen für die Listenfelder ausgelesen werden, die aktualisiert werden sollen. Die Zusatzinformationen sind gespeichert in **`oFeld.Tag`**. Die Bezeichnungen der Listenfelder werden anhand des Kommas voneinander getrennt und in das bereits definierte Array eingelesen: **`arList = Split(stTag, ",")`**. Hier wird davon ausgegangen, dass die Listenfeldbezeichnungen nur durch ein Komma getrennt werden. Sonst müssten gegebenenfalls noch Leerstellen mit Hilfe von **`Trim()`** entfernt werden.

Des weiteren erfolgt noch ein Zugriff auf das Formular, das die gefilterten Daten darstellen soll.

Das Listenfeld gibt seinen Wert ab (**`commit()`**). Die Änderung in dem Formular, in dem das Listenfeld steht, wird abgespeichert.

Anschließend wird in einer Schleife das Array durchgegangen. **`LBound(arList())`** zeigt den unteren Wert '0' des Arrays an. **`Ubound(arList())`** verweist auf den höchsten Wert, im praktischen Beispiel '1'.

Jede Bezeichnung für ein Listenfeld wird eingelesen. So präsentiert **`arList(0)`** das erste Listenfeld aus den Zusatzinformationen, aus dem obigen Screenshot also lboFilter2, **`arList(1)`** ist gleich lboFilter3. Das entsprechende Feld wird durch **`refresh()`** neu eingelesen.

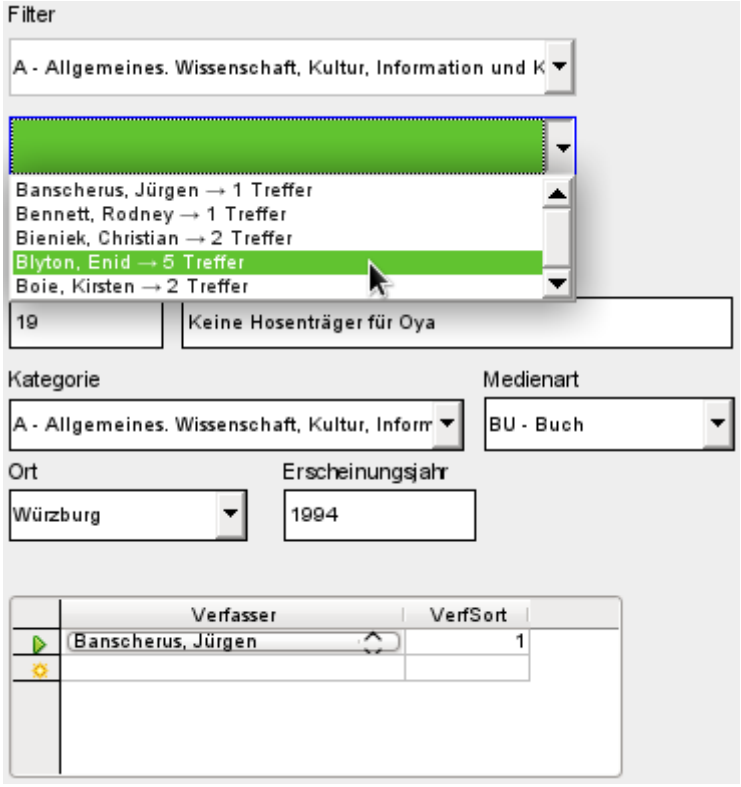
Schließlich wird noch nachgesehen, ob der Wert für das aktuelle Listenfeld, den Auslöser für das Ereignis, jetzt leer ist. Nur dann soll auch das aktuelle Listenfeld neu eingelesen werden. Ansonsten kann einfach ein anderer Wert aus der alten Liste gewählt werden, falls der erste Wert nicht zum Erfolg führte.

Schließlich wird noch das Formular, das das Ergebnis der Filterung darstellen soll, neu geladen.

Das Formular «Filter_Formular_Subformular_bedingende_Filter_Datensaetze»

Dieses Formular hat lediglich im Vergleich zu den vorhergehenden Formularen andere Datengrundlagen für die Listenfelder. Hierfür wurden die Abfragen «Listenfeld_Kategorie_bedingt_Datensaetze», «Listenfeld_Verfasser_bedingt_Datensaetze_optimiert» und «Listenfeld_Medienart_bedingt_Datensaetze» erstellt.

Das Makro «Filter_bedingt» ist an jedes Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.



Neben den noch auswählbaren Verfassern nach der Vorauswahl der Kategorie 'A – Allgemeines …' ist aufgezeigt, wie viele Datensätze noch übrig bleiben würden, wenn der entsprechende Verfasser ausgewählt würde. Wird im obigen Beispiel die Verfasserin Enid Blyton ausgewählt, so bleiben noch 5 Datensätze in dem unten dargestellten gefilterten Hauptformular zur Auswahl.

Durch die Vorauswahl der Kategorie wurde der Inhalt des Listenfeldes für die Verfasser bereits stark eingeschränkt. Ist diese Vorauswahl nicht gegeben, so bremst gerade dieses Listenfeld durch die entsprechend komplizierte Abfrage und den entsprechenden Datenumfang das Formular recht stark aus.

Das Formular «Suche_Formular»

Dieses Formular unterscheidet sich bis auf den fehlenden Aktualisierungsbutton nicht von dem in der Datenbank «Beispiel_Suchen_und_Filtern_ohne_Makros». Der Aktualisierungsvorgang wird durch die Einbindung des Makros «Filter_Suche» an das Eingabefeld für den Suchbegriff über **Eigenschaften:Textfeld → Ereignisse → Bei Fokusverlust** erzeugt. Der Fokusverlust wurde hier statt «Text modifiziert» gewählt, weil eine Modifikation des Textes jede einzelne Buchstabeneingabe ist, das Makro also beim Tippen laufend ausgelöst würde.

Markosteuerung für die Aktualisierung des Formulars

Mit diesem Makro kann sowohl ein Suchformular als auch ein Filterformular betrieben werden. Es ist also statt des Makros «Filter» auch bei einem Filter einsetzbar. Über das auslösende Formularfeld wird der neue Wert geschrieben und das Formular für die Anzeige des Filter- bzw. Suchergebnisses neu eingestellt.

```
SUB Filter_Suche(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm2 = oDrawpage.forms.getByName("MainForm")
```

```
        oFeld = oEvent.Source.Model
        oForm1 = oFeld.Parent
        oFeld.commit()
        oForm1.updateRow()
        oForm2.reload()
    END SUB
```

Der Kontakt zum Formular, das das Suchergebnis darstellen soll, wird auf die übliche Weise über die **`Drawpage`** erstellt.

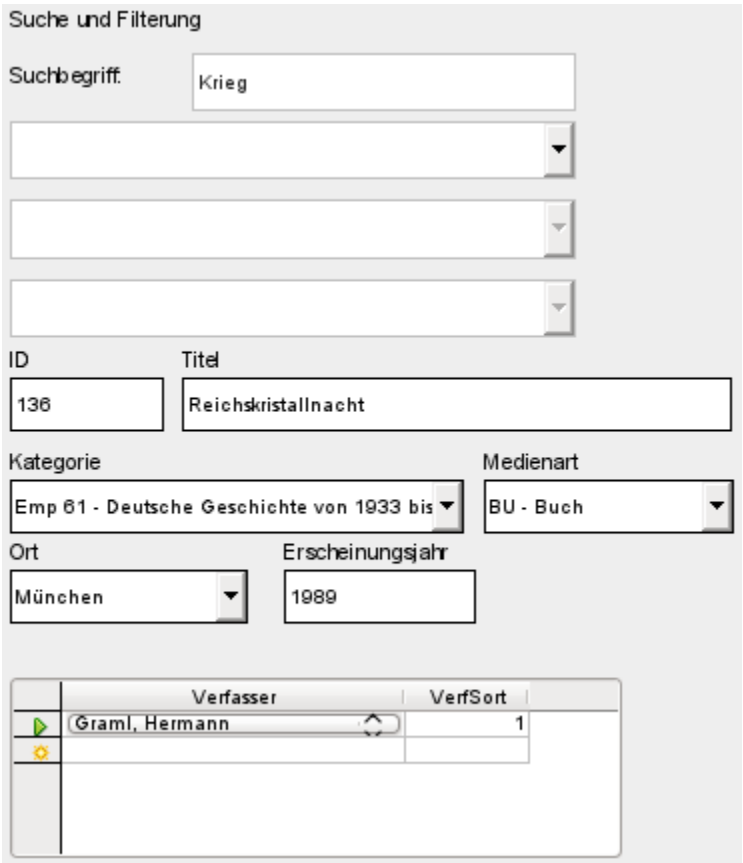
Der Kontakt zum Filterformular und zu dem Eingabefeld entsteht durch das auslösende Ereignis. Dadurch wird das Feld und das Formular ermittelt. Mit **`commit()`** wird der Wert in die Tabelle "Filter" übertragen, mit **`updateRow()`** wird der geänderte Datensatz gespeichert. Anschließend wird das Formular zur Darstellung des Suchergebnisses über **`reload()`** neu geladen.

Das Formular «Suche_Formular_Subformular»

Lediglich die Datenquelle des Hauptformulars unterscheidet sich in diesem Formular von dem Formular «Suche_Formular». Das Makro sowie die Einbindung des Makros in das Formular sind identisch.

Das Formular «Suche_Filter_hierarchisch_allround»

In diesem Formular sind Suchfunktion und Filter miteinander kombiniert. Hinzu kommt, dass die Filter hierarchisch angeordnet sind. Der zuerst gewählte Filter beeinflusst die nachfolgenden Filter. Es steht aber vorher bereits fest, welcher Filter zuerst gewählt werden darf. Die anderen Listenfelder sind vorher deaktiviert.

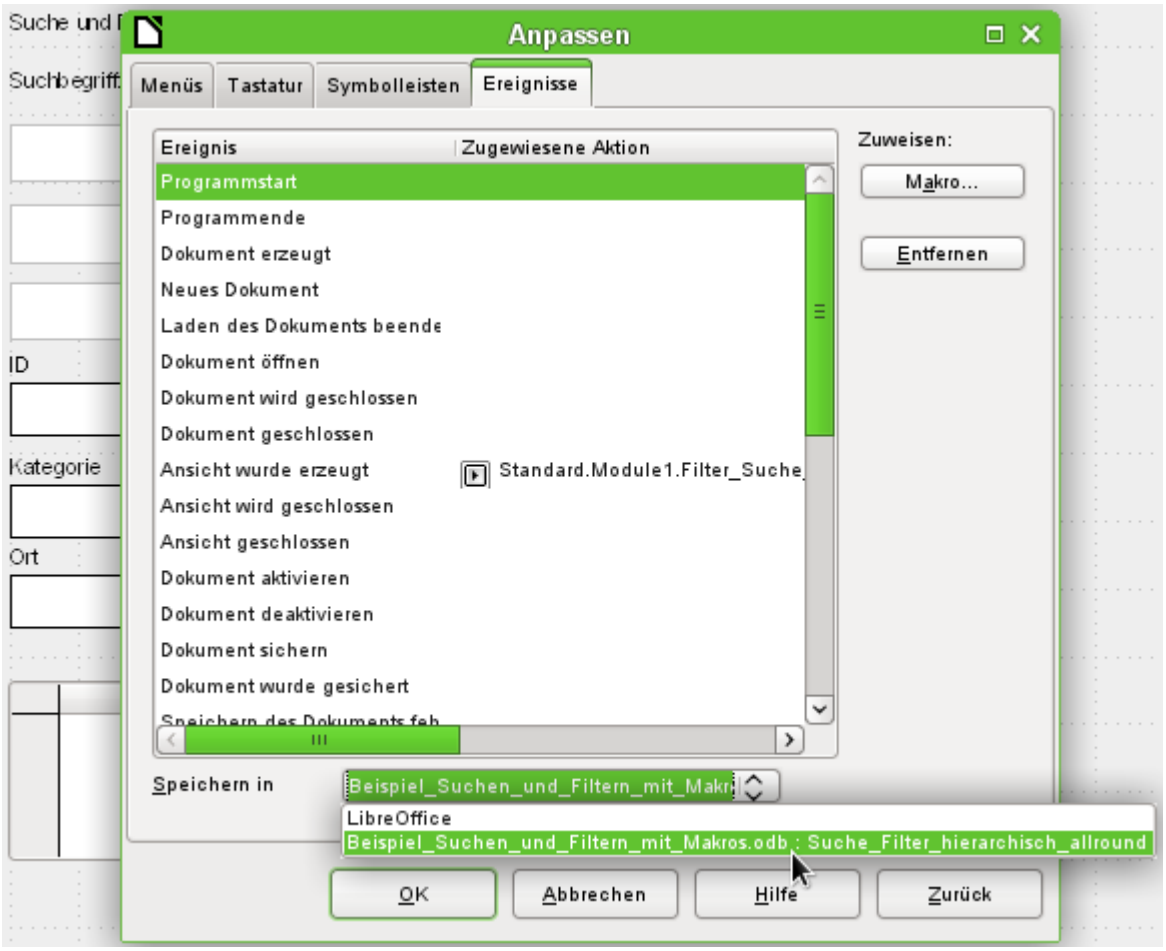


Hier ist bereits ein Suchbegriff eingegeben. Das erste Listenfeld ist bereits von Anfang an aktiviert, erkennbar an dem schwarzen nach unten zeigenden Dreieck. Listenfeld 2 und Listenfeld 3 sind an der entsprechenden Stelle ausgegraut.

Das Formular muss zur Einhaltung der Hierarchie zumindest in dem Inhalt der Listenfelder 2 und 3 leer sein. Das bedeutet, dass in der Tabelle "Filter" keine Werte für diese Listenfelder eingetragen werden sein dürfen. Über ein Makro hier gleich komplett beim Start des Formulars der eventuell bestehende Inhalt der Filtertabelle gelöscht.

Die Datenquellen für die Listenfelder wurden durch die Abfragen «Listenfeld_Kategorie_bedingt_Suche_Filter», «Listenfeld_Verfasser_bedingt_Suche_Filter» und «Listenfeld_Medienart_bedingt_Suche_Filter» erstellt.

Das Makro «Filter_Suche_hierarchisch_Start» muss an den Aufbau des Formulardokumentes gebunden werden: **Extras → Anpassen → Ereignisse → Ansicht wurde erzeugt**. Nur so wird es ausgeführt bevor ein Inhalt in die Formularfelder eingelesen wird.



Die Makroverbindung wird hier in dem Formular «Suche_Filter_hierarchisch_allround» gespeichert.

Das Makro «Filter_Suche» wird an das Textfeld zum Suchbegriff gebunden: **Eigenschaften:Textfeld → Ereignisse → Bei Fokusverlust**.

Das Makro «Filter_Suche_hierarchisch» ist an jedes Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.

In den Zusatzinformationen des ersten Listenfeldes lboFilter1 wird «lboFilter2,lboFilter3» eingetragen. In den Zusatzinformationen von lboFilter2 steht nur «lboFilter3». Die Zusatzinformationen von lboFilter3 sind komplett leer.

Markosteuerung für den Start des Formulars

```
SUB Filter_Suche_hierarchisch_Start
   DIM oDatenquelle AS OBJECT
```

```
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSql AS STRING
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "DELETE FROM ""Filter"""
    oSQL_Anweisung.executeUpdate(stSql)
    stSql = "INSERT INTO ""Filter"" (""ID"") VALUES (True)"
    oSQL_Anweisung.executeUpdate(stSql)
END SUB
```

Zuerst wird die Datenquelle für das aktuelle Datenbankdokument aufgesucht. Interne Formulare einer Base-Dokumentes erzeugen beim Öffnen automatisch eine Verbindung zu der Datenquelle. Besteht bisher keine Verbindung, so wird anschließend eine Verbindung zur Datenbank hergestellt.

Jede SQL-Anweisung muss über **`createStatement()`** zuerst vorbereitet werden. Der Inhalt für die Anweisung wird anschließend in einer Variablen gespeichert. Anschließend wird die Variable mit dem Befehl **`executeUpdate()`** an die Datenbank weiter gegeben. Eine Ausgabe der Daten wird nicht erwartet.

Mit **`DELETE FROM "Filter"`** werden alle Datensätze in der Tabelle "Filter" gelöscht. Im Makro sind hier doppelte Anführungszeichen um den Tabellennamen gesetzt, damit die einfachen Anführungszeichen in dem Kommando weiter gegeben werden. Anführungszeichen werden mit Anführungszeichen maskiert.

Mit **`INSERT INTO "Filter" ("ID") VALUES (TRUE)`** wird der für die Filterung benötigte Datensatz wieder erstellt. Einziger Eintrag ist hier der Wert für das Primärschlüsselfeld, der bei der Filterung existieren muss.

Markosteuerung für die hierarchische Filterung des Formulars

```
SUB Filter_Suche_hierarchisch(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM stTag AS String
    DIM arList()
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeld = oEvent.Source.Model
    oFeld.commit()
    oForm1 = oFeld.Parent
    oForm1.updateRow()
    stTag = oFeld.Tag
    arList = Split(stTag, ",")
    FOR i = LBound(arList()) TO UBound(arList())
       IF IsEmpty(oFeld.currentValue) THEN
          oForm1.getByName(arList(i)).Enabled = FALSE
       ELSE
          oForm1.getByName(arList(0)).Enabled = TRUE
       END IF
       oForm1.getByName(arList(i)).BoundField.UpdateNULL()
       oForm1.updateRow()
       oForm1.getByName(arList(i)).refresh()
    NEXT
    oForm2.reload()
END SUB
```

Die Verbindungen zu den Formularen und zum aktuellen Formularfeld werden wie bereits in den vorhergehenden Makros erstellt. Der Inhalt aus den Zusatzinformationen (**`Tag`**) wird in das Array «arList» eingelesen. Anschließend wird das Array durchlaufen.

Wenn das leere Feld gewählt wurde, dann wird für jeden Eintrag in der Arrayliste das Listenfeld deaktiviert: **`Enabled = FALSE`**. Wenn das gewählte Feld nicht leer ist wird das erste Feld aus dem Array aktiviert: **`Enabled = TRUE`**.

Für alle Listenfelder aus dem Array wird der Datenbestand entfernt: **`UpdateNull()`**. Danach wird der Datensatz gespeichert und das Listenfeld neu eingelesen, also ohne Wert dargestellt. So kann mit dem aktuellen Listenfeld auch eine gerade erstellte Wahl wieder geändert werden, ohne dass dadurch plötzlich kein Datensatz mehr in der Ergebnismenge vorhanden ist.

Das auslösende Feld wird neu eingelesen und zeigt damit nur noch einen Datensatz an. Schließlich wird das zu filternde Formular neu geladen.

Das Formular «Suche_Filter_hierarchisch_allround_Datensaetze»

Äußerlich ist das Formular gleich aufgebaut wie das Formular «Suche_Filter_hierarchisch_allround». Die Listenfelder zeigen allerdings neben den Feldnamen die zu erwartenden Datensätze bei einer weiteren Filterung an. Dazu mussten wieder entsprechende Abfragen erstellt werden, die weiter oben erwähnt sind.

Besonderes Merkmal ist hier, dass das Formular trotz der Anzeige der Datensätze zügig startet. Dies liegt daran, dass zuerst nur das erste Listenfeld mit einer Datenquelle verbunden ist. In dem ersten Listenfeld steht in den Zusatzinformationen jetzt allerdings nicht nur «lboFilter2,lboFilter3», sondern «lboFilter2>Listenfeld_Kategorie_bedingt_Suche_Filter_Datensaetze,lboFilter3». Neben den Bezeichnungen für die anderen Listenfeldern ist dort also auch die Datenquelle für das nächste Listenfeld verzeichnet. Diese Datenquelle wird über ein Makro mit dem entsprechenden Listenfeld verbunden und anschließend das Listenfeld neu eingelesen.

Die Listenfelder lboFilter2 und lboFilter3 haben zwar unter **Eigenschaften:Listenfeld → Daten → Art des Listeninhaltes → Abfrage** stehen. Das Feld für den Listeninhalt selbst ist aber leer.

Das Makro «Filter_Suche_hierarchisch_allround» ist an jedes Listenfeld gebunden: **Eigenschaften:Listenfeld → Ereignisse → Modifiziert**.

Markosteuerung für die hierarchische Filterung des Formulars

```
SUB Filter_Suche_hierarchisch_allround(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM stTag AS String
    DIM arList()
    DIM arTmp()
    DIM stSql(0) AS STRING
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm2 = oDrawpage.forms.getByName("MainForm")
    oFeld = oEvent.Source.Model
    oFeld.commit()
    oForm1 = oFeld.Parent
    oForm1.updateRow()
    stTag = oFeld.Tag
    arList = Split(stTag, ",")
    FOR i = LBound(arList()) TO UBound(arList())
        IF i = 0 THEN
            arTmp = Split(arList(0), ">")
            arList(0) = arTmp(0)
```

```
            stSql(0) = arTmp(1)
        END IF
        IF IsEmpty(oFeld.currentValue) THEN
            stSql(0) = ""
            oForm1.getByName(arList(i)).Enabled = FALSE
            oForm1.getByName(arList(i)).ListSource = stSql
        ELSE
            oForm1.getByName(arList(0)).Enabled = TRUE
            oForm1.getByName(arList(0)).ListSource = stSql
        END IF
        oForm1.getByName(arList(i)).BoundField.UpdateNULL()
        oForm1.updateRow()
        oForm1.getByName(arList(i)).refresh()
    NEXT
    oForm2.reload()
END SUB
```

Der Grundaufbau dieser Prozedur entspricht erst einmal der Prozedur «Filter_Suche_hierarchisch». Bereits bei der Deklaration kommen zwei Arrays als Variablen hinzu. Das Array **`arTmp()`** nimmt nur zwischendurch das Ergebnis der Funktion **`Split()`** auf und gibt die Teile entsprechend weiter. Das Array **`stSql()`** wird von vornherein als ein Array definiert, das nur einen Datensatz enthält, der außerdem ein **`STRING`** sein muss.

Die Inhalte aus den Zusatzinformationen werden, wie bei «Filter_Suche_hierarchisch», ausgelesen und in einer Schleife abgearbeitet. Nur beim ersten Durchgang der Schleife, wenn also **`i = 0`**, wird das erste Element des Arrays **`arList()`** erneut gesplittet, jetzt aber bei dem Zeichen **`«>»`**. Der erste Teil enthält dann den Namen des Listenfeldes, z.B. 'lboFilter2'. Der zweite Teil enthält den Namen der Abfrage, die das Listenfeld mit den auszuwählenden Daten versehen soll. Der zweite Teil ist der einzige Eintrag, der in das Array **`stSql`** als **`stSql(0)`** eingetragen wird.

Ist der ausgewählte Inhalt aus dem Listenfeld leer, so soll auch der Name der Datenquelle wieder gelöscht werden. Daher steht hier **`stSql(0) = ""`**. Ein Listenfeld, das anschließend deaktiviert wird, braucht nicht den gesamten Datenbestand wieder einzulesen. Mit **`ListSource = stSql`** wird das gesamte Array als Datenquelle angegeben. **`ListSource`** erwartet ein Array vom Typ **`STRING`** – als Hinweis auf eine Abfrage, als Abfrage direkt, als Datensammlung …

Ist ein Inhalt in dem aktuellen Listenfeld ausgewählt worden (**`oFeld.currentValue`** ist nicht leer), so wird das in der Hierarchie folgende Listenfeld aktiviert sowie mit einer Datenquelle versehen.

Standardmäßig werden alle dem aktuellen Listenfeld in der Hierarchie folgenden Listenfelder auf **`NULL`** gesetzt, der Filter mit **`updateRow()`** auf die entsprechenden Werte eingestellt und die folgenden Listenfelder neu eingelesen.

Zum Schluss wird der entsprechende Filterwert auf das Formular, das das Filterergebnis anzeigen soll, übertragen: **`oForm2.reload()`**.

Aktuelle Standardwerte für Datum und Zeit setzen

In Formularen gibt es für Formularfelder die Möglichkeit, Standardwerte zu setzen. Damit werden über die grafische Benutzeroberfläche Wertvorgaben für bestimmte Felder gemacht, die abgespeichert werden, wenn sie nicht überschrieben werden.

Auch bei der Erstellung von Tabellen ist in den Feldeigenschaften ein Feld für den «Defaultwert» vorgesehen. Ein Eintrag an dieser Stelle erzeugt ebenfalls einen festen Vorgabewert für die Eingabe in ein Feld – jetzt aber bei Tabellen. Er hat im übrigen nichts mit dem Default-Wert der eigentlichen Datenbank zu tun. Der Default-Wert, definiert in einer Datenbank, wird nur dann geschrieben, wenn ein Feld als leeres Feld abgespeichert werden soll.

Alle Standardwerte der grafischen Benutzeroberfläche haben den Nachteil, dass sie nicht flexibel sind. Es kann nicht das beim Öffnen des Formulars aktuelle Datum oder die aktuelle Zeit eingegeben werden.

Die folgende Datenbank zeigt an Beispielen, wie es möglich ist, ein aktuelles Datum bzw. eine aktuelle Datum-Zeit-Kombination beim Erstellen von Datensätzen zu erzeugen. Es zeigt außerdem, welche Möglichkeiten es gibt, ein Datum hinterher mit möglichst geringem Aufwand ändern zu lassen. Typische Beispiel wären hier eine Zeitmessung in Form von Arbeitsbeginn und Arbeitsende oder ein Vermerk, wann ein Datensatz erstellt und wann er zuletzt geändert worden ist.

Tabellen

Die Tabellen stellen nur einfache Beispieltabellen dar. Sie sollen nur zeigen, wie aktuelle Standardwerte für ein Datum bzw. ein Datum mit Zeitangabe gesetzt werden können.

In der Tabelle "Datum_Aenderdatum" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Name	Text	Der Name einer Person oder irgendeine Bezeichnung. Steht hier nur als Beispieleintrag ohne weitere Bedeutung.
Datum	Datum	Hier soll ein Startdatum gespeichert werden, das nachher nicht mehr verändert wird.
AenderDatum	Datum	Hier soll beim Erstellen des Datensatzes das gleiche Datum wie im Feld "Datum" abgespeichert werden. Dieses Feld wird beim erneuten Aufrufen des Datensatzes aber gegebenenfalls überschrieben.

Die Tabelle "Datum_SQLDefault_Aenderdatum" ist von den Feldern her völlig gleich aufgebaut. Hier ist allerdings über **Extras → SQL** hinterher ein SQL-Defaultwert eingestellt worden:

```
ALTER TABLE "Datum_SQLDefault_Aenderdatum" ALTER COLUMN "Datum" SET
DEFAULT TODAY;
ALTER TABLE "Datum_SQLDefault_Aenderdatum" ALTER COLUMN "AenderDatum"
SET DEFAULT TODAY;
```

Damit werden in dieser Tabelle beim Abspeichern eines neuen Datensatzes die Felder "Datum" und "AenderDatum" automatisch mit dem aktuellen Datum versehen, sofern nicht ein anderes Datum eingetragen wurde.

In der Tabelle "Zeitstempel_Aenderstempel" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Name	Text	Der Name einer Person oder irgendeine Bezeichnung. Steht hier nur als Beispieleintrag, z.B. zur Ermittlung von Arbeitszeit.
Zeitstempel	Datum/Zeit	Hier soll eine Startzeit zusammen mit Datum gespeichert werden (**`TIMESTAMP`**), die nachher nicht mehr verändert wird.

Zeitstempel_Aenderung	Datum/Zeit	Hier soll beim Erstellen des Datensatzes die gleiche Zeit mit Datum wie im Feld "Zeitstempel" abgespeichert werden. Dieses Feld wird beim erneuten Aufrufen des Datensatzes aber gegebenenfalls überschrieben.

Die Tabelle "Zeitstempel_SQLDefault_Aenderstempel" ist von den Feldern her völlig gleich aufgebaut. Hier ist allerdings über **Extras → SQL** hinterher ein SQL-Defaultwert eingestellt worden:

```
ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel" ALTER COLUMN
"Zeitstempel" SET DEFAULT NOW;
ALTER TABLE "Zeitstempel_SQLDefault_Aenderstempel" ALTER COLUMN
"Zeitstempel_Aenderung" SET DEFAULT NOW;
```

Damit werden in dieser Tabelle beim Abspeichern eines neuen Datensatzes die Felder "Zeitstempel" und "Zeitstempel_Aenderung" automatisch mit der aktuellen Zeit und dem aktuellen Datum versehen, sofern nicht eine andere Zeit und ein anderes Datum eingetragen wurde.

Abfragen

Die Datenbank enthält drei Abfragen. Für die Eingabe des Standarddatums ist lediglich die Abfrage «DatumZeitvorgabe» von Bedeutung. Die weiteren Abfragen dienen als Berichtsgrundlage («Zeitdifferenzen») oder als Vorlage für ein Formular, in dem gleichzeitig berechnete Differenzen betrachtet werden sollen («Zeitdifferenzen_Formularvorlage»).

DatumZeitvorgabe

Die Abfrage «DatumZeitvorgabe» dient lediglich dazu, das aktuelle Datum und den aktuellen Zeitstempel von der Datenbank zu erfragen. Damit kann ohne weitere Zuhilfenahme von direkter SQL-Eingabe oder Makros das aktuelle Datum und (etwas begrenzt) auch der Zeitstempel in ein Formular übertragen werden.

```
SELECT DISTINCT CURRENT_TIMESTAMP AS "Jetzt", CURRENT_DATE AS "Heute"
FROM "Datum_Aenderdatum"
```

Für eine Abfrage muss immer eine Tabelle als Basis stehen. Dies ist hier die Tabelle "Datum_Aenderdatum". In diesem Fall spielt es keine Rolle, welche Tabelle das ist, da nur Standardwerte aus der Datenbank ermittelt werden sollen: das aktuelle Datum **`CURRENT_DATE`** und die aktuelle Zeit kombiniert mit dem Datum **`CURRENT_TIMESTAMP`**. Durch den Zusatz **`DISTINCT`** werden die Werte nur einmal dargestellt. Ansonsten würde die Datenbank so viele Datensätze wiedergeben, wie die Tabelle bereits an Datensätzen hat – mit lauter gleichen Datums- und Zeitstempelangaben.

Zeitdifferenzen

Zur Auswertung von Zeiteingaben in einem Bericht wurde die Abfrage «Zeitdifferenzen» erstellt.

```
SELECT "ID", "Name", "Zeitstempel", "Zeitstempel_Aenderung",
   CONVERT ( YEAR( "Zeitstempel" ) || '-' ||
      RIGHT( '0' || MONTH( "Zeitstempel" ), 2 ) || '-' ||
      RIGHT( '0' || DAY( "Zeitstempel" ), 2 ) , DATE )
      AS "StartDatum",
   DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" )
      AS "Differenz_Minuten"
FROM "Zeitstempel_Aenderstempel"
```

Zuerst wird die komplette Tabelle "Zeitstempel_Aenderstempel" ausgewählt. Das Feld "ID" ist dabei nur notwendig, wenn so eine Abfrage für die Eingabe von Daten benutzt werden soll. Um eine Abfrage editierbar zu halten muss der Primärschlüssel der in der Abfrage enthaltenen Tabellen ebenfalls enthalten sein.

Aus dem Feld "Zeitstempel" ließe sich mit Hilfe der Formatierung auch lediglich das Datum darstellen. In der Abfrage wird zu Demonstrationszwecken gezeigt, wie die Datumsdarstellung auch erreicht werden kann:

Tag, Monat und Jahr können nur über separate Abfragen des Feldes "Zeitstempel" ermittelt werden. Um auch bei einstelligen Tageszahlen und Monatszahlen die vorangestellte '0' zu erhalten werden eine '0' und der ermittelte Tag zusammengefasst (**`'0' || DAY( "Zeitstempel" )`**) und dann aus dem entstehenden Text von rechts aus die 2 übrigbleibenden Zeichen ausgelesen. Das Datum wird dann in einem für SQL lesbares Datumsformat geschrieben: vierstellige Jahreszahl, zweistellige Monatszahl und zweistellige Tageszahl, verbunden mit einem Bindestrich (**`yyyy-mm-dd`**). Damit ist die Abfrage auch für andere Datumsschreibweisen nutzbar, denn dieser Text wird jetzt in das Datumsformat der Datenbank über **`CONVERT ( Datumstext, DATE )`** umgewandelt. Dadurch kann die Ausgabe z.B. im Bericht wieder entsprechend nach lokalen Einstellungen formatiert werden.

Mit **`DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" )`** wird die Zeitdifferenz zwischen der Startzeit und der Endzeit in Minuten ermittelt. Diese Zeitdifferenz kann später im Bericht addiert werden und gegebenenfalls auch durch Umrechnung als eine Angabe von Minuten und Stunden erfolgen.

Zeitdifferenzen_Formularvorlage

Als Formularvorlage ist die wesentlich umfangreichere Abfrage «Zeitdifferenzen_Formularvorlage» gedacht.

```
SELECT "ID", "Name", "Zeitstempel", "Zeitstempel_Aenderung",
   CONVERT ( YEAR( "Zeitstempel" ) || '-' ||
      RIGHT( '0' || MONTH( "Zeitstempel" ), 2 ) || '-' ||
      RIGHT( '0' || DAY( "Zeitstempel" ), 2 ) , DATE )
      AS "StartDatum",
   ( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) -
      MOD( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ),
      60 ) ) / 60 || ':' || RIGHT( '0' || MOD( DATEDIFF( 'mi',
      "Zeitstempel", "Zeitstempel_Aenderung" ), 60 ), 2 )
      AS "Differenz_Stunden",
   ( ( SELECT SUM( DATEDIFF( 'mi', "Zeitstempel",
      "Zeitstempel_Aenderung" ) ) FROM "Zeitstempel_Aenderstempel"
      WHERE "Name" = "a"."Name" ) - MOD( ( SELECT SUM( DATEDIFF( 'mi',
      "Zeitstempel", "Zeitstempel_Aenderung" ) ) FROM
      "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ),
      60 ) ) / 60 || ':' || RIGHT( '0' || MOD( ( SELECT SUM( DATEDIFF(
      'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) ) FROM
      "Zeitstempel_Aenderstempel" WHERE "Name" = "a"."Name" ), 60 ), 2
      ) AS "Gesamt_Stunden"
FROM "Zeitstempel_Aenderstempel" AS "a"
```

Bis zur Abfrage des Startdatums ist diese Abfrage gleich der Abfrage «Zeitdifferenzen»: Danach wird beständig die Zeitdifferenz in Minuten genutzt: **`DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" )`**. Die Zahlenangabe kann nicht in ein Zeitformat der Datenbank umgewandelt werden, da in der Datenbank nur Zeiten bis maximal 24 Stunden verwaltet werden. Es wird hier also lediglich in einer Textform die Zeitdarstellung nachgebildet.

Zuerst werden die Minuten durch 60 dividiert, um die Anzahl in Stunden zu ermitteln. Eigentlich müsste es hier genügen, beim Ergebnis die Nachkommastellen abzuschneiden oder auch abzurunden. Dies gelingt leider nicht, da bei den Befehlen TRUNCATE() und CEILING() das Ergebnis immer als Dezimalzahl mit einer Nachkommastelle ausgegeben wird – auch wenn die Nachkommastelle 0 ist. Um ein Ergebnis ohne Nachkommastellen zu erhalten muss von vornherein der ver-

bleibende Rest der Division **`MOD ( Minuten, 60)`** subtrahiert werden. Dann wird bei den anschließenden Division die korrekte Ganzzahl ausgegeben.

Anschließend wird ein Doppelpunkt über **`||':'`** angehängt und an diesen wiederum der Rest aus der Division der Minuten durch 60. Vor die Division wird eine führende '0' gesetzt, damit auf jeden Fall eine zweistellige Zahl für die Minutendarstellung vorhanden ist. Wie beim Datum werden mit **`RIGHT (Text,2)`** einfach die zwei Zeichen des Textes übernommen, die am weitesten rechts stehen.

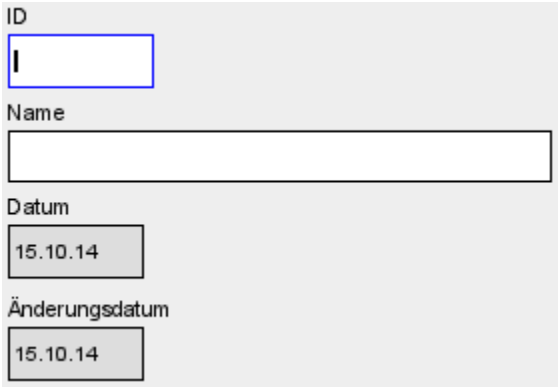
Mit dem entsprechenden Verfahren wird auch die Gesamtzahl an Stunden ermittelt. Da diese pro Person ermittelt werden soll sind hier für die Ermittlung der Stunden und Minuten korrelierende Unterabfragen notwendig. Dazu ist zuerst einmal der Tabelle der Hauptabfrage "Zeitstempel_Aenderung" der **`Alias "a"`** zugeordnet. Anschließend wird in der Unterabfrage die Summe **`SUM`** für die Zeitdifferenz in Minuten **`( DATEDIFF( 'mi', "Zeitstempel", "Zeitstempel_Aenderung" ) )`** gebildet – aber nur bezogen auf den jeweiligen Namen, der im aktuellen Datensatz der Hauptabfrage steht: **`WHERE "Name" = "a"."Name"`**. Die Berechnungen und Umwandlungen der Ergebnisse aus den korrelierenden Unterabfragen sind gleich den Berechnungen und Umwandlungen aus der direkten Ermittlung der Zeitdifferenz in Minuten für den jeweiligen Datensatz. Das Ergebnis wird als "Gesamt_Stunden" ausgegeben.

Diese Abfrage funktioniert aus nicht näher untersuchten Gründen nicht mit LO 4.1.

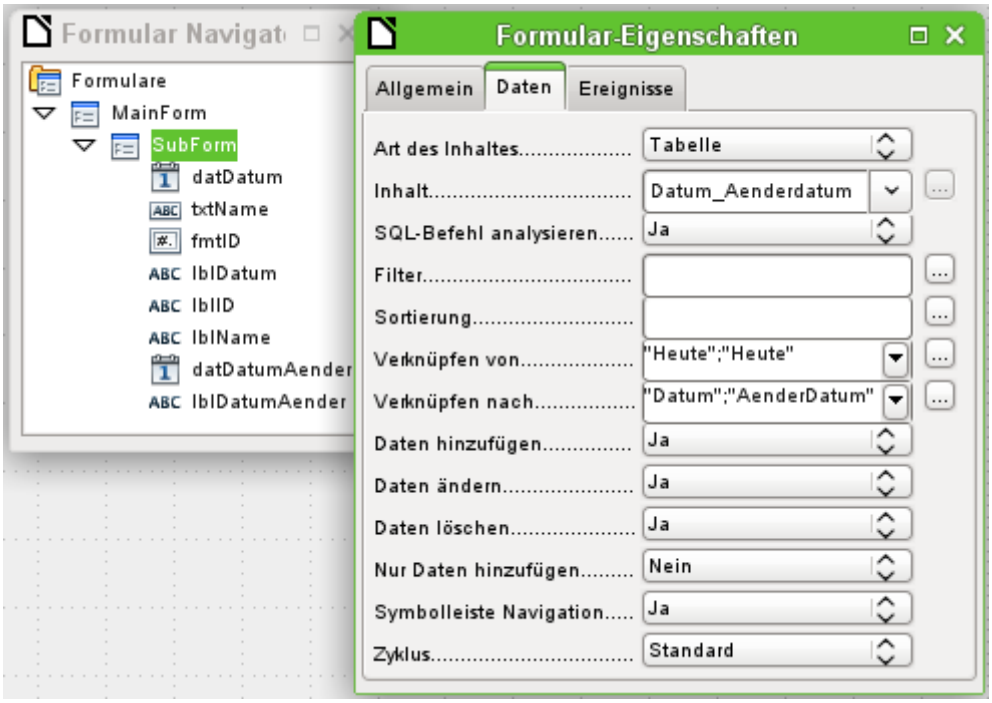
Formulare

Aktuelles Datum oder aktuelle Zeit über Abfrage einsetzen

Die Formulare «Defaultdatum_Subform_neue_Datensaetze» und «Defaultdatum_Zeit_Subform_neue_Datensaetze» arbeiten nach dem gleichen Prinzip. Wird das Formular geöffnet, so ist bereits das Datum bzw. der Zeitstempel vorgegeben. Im Datumsformular erscheinen die Datensätze des aktuellen Tages und können auch nur Datensätze für den aktuellen Tag eingegeben werden. Im Zeitformular erscheinen auf keinen Fall die vorhergehenden Datensätze, da die Zeit zum Öffnen des Formulars maßgebend ist.

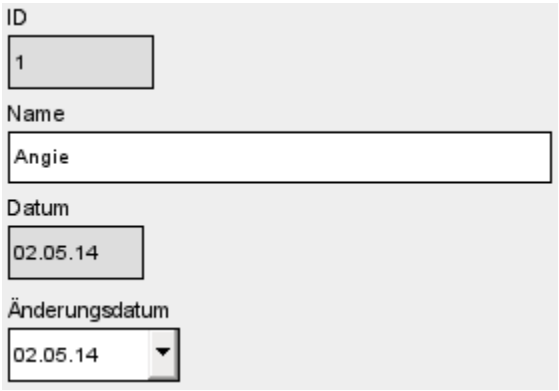


Es ist lediglich das Datenfeld für den Primärschlüssel "ID" sowie für das Textfeld "Name" ausfüllbar. Die in Grau gehaltenen Formularfelder sind schreibgeschützt.



Wird das Formular zum Bearbeiten geöffnet und der Formularnavigator aufgerufen, so wird die Verbindung von Hauptformular **`MainForm`** und Unterformular **`SubForm`** sichtbar. Das Hauptformular hat als Inhalt die Abfrage «DatumZeitvorgabe». Von dem Hauptformular wird das ermittelte aktuelle Datum an das Unterformular übergeben. Das Feld "Heute" ist mit den Feldern "Datum" und "AenderDatum" des Unterformulars verknüpft.

Im Hauptformular des Datumformulars existieren keine Formularfelder. Im Hauptformular des Zeitformulars ist ein Button enthalten. Dieser Button ist in den **Eigenschaften: Schaltfläche → Allgemein → Aktion** auf **Formular aktualisieren** eingestellt. Dadurch werden die Zeiten gegebenenfalls neu eingelesen und übertragen. Würde dies nicht möglich sein, so hätten alle nach dem öffnen des Formulars erstellten Datensätze den gleichen Zeitstempel.



Um ein Datum bzw. einen Zeitstempel zu ändern ist ein separates Formular erforderlich, hier das Formular «Defaultdatum_Aenderung». Die Änderung kann nicht in dem vorhergehenden Formular erfolgen, da das Änderungsfeld dort mit dem Hauptformular verbunden ist und bereits eine Datums- bzw. Zeitvorgabe hat.

Mit Hilfe der Hauptformular-Unterformular-Konstruktion kann auf einfache Weise das aktuelle Datum übertragen und automatisch mit Datensätzen abgespeichert werden. Bei Zeitangaben muss aber schon eine laufende Aktualisierung durch einen Button erfolgen. Änderungen von bereits eingegebenen Datums- bzw. Zeitstempelwerten sind nur über die direkte Eingabe per Hand möglich.

Aktuelles Datum oder aktuelle Zeit über SQL-Default-Wert

Die Formulare «Defaultdatum_Makro_SQL» und «Defaultdatum_Zeit_makro_SQL» sind wiederum nach dem gleichen Prinzip aufgebaut.

Bei diesen Formularen erscheint beim Erstellen eines neuen Datensatzes kein Eintrag in den Datums- bzw. Zeitstempel-Feldern. Die Tabelle, die mit dem Formular bearbeitet wird, hat einen Defaultwert über SQL für die Datums- bzw. Zeitstempel-Felder erhalten. Dieser Wert wird beim Abspeichern eines neuen Datensatzes in die Tabelle geschrieben. Der Nutzer des Formulars sieht den Wert also erst, wenn ein Datensatz abgespeichert wurde.

Wenn ein Datensatz geändert werden soll, so kann ein neues Datum bzw. ein neuer Zeitstempel über den Button erstellt werden. Der Button ist hierfür über das **Ereignis → Aktion ausführen** einem Makro verbunden.

Makro für einen neuen Zeitstempel über SQL

In den Zusatzinformationen das Buttons ist der SQL-Code angegeben, der bei einer Änderung ausgeführt werden soll. Außerdem ist hinter einem Semikolon vermerkt, in welchen der Formularfelder der Wert für den Primärschlüssel steht. Der Primärschlüssel wird für die Änderung nur des einen Datensatzes benötigt. Ohne den Schlüssel würden Änderungen für alle Datensätze gelten.

Eintrag für den Zeitstempel-Button:

```
UPDATE "Zeitstempel_SQLDefault_Aenderstempel" SET
"Zeitstempel_Aenderung" = NOW;fmtID
```

Eintrag für den Datumsstempel-Button:

```
UPDATE "Datum_SQLDefault_Aenderdatum" SET "AenderDatum" = NOW;fmtID
```

Das Makro ist an den auslösenden Button gebunden.

```
SUB Update(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeldID AS OBJECT
    DIM stTag AS STRING
    DIM arList()
    oFeld = oEvent.Source.Model
    oForm = oFeld.Parent
    stTag = oFeld.Tag
    arList = Split(stTag, ";")
    oFeldID = oForm.getByName(Trim(arList(1)))
    IF NOT IsEmpty(oFeldID.currentValue) THEN
        oDatenquelle = ThisComponent.Parent.CurrentController
        If NOT (oDatenquelle.isConnected()) THEN
            oDatenquelle.connect()
        END IF
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        stSql = arList(0) + " WHERE ""ID"" = " + oFeldID.CurrentValue
```

```
            oSQL_Anweisung.executeUpdate(stSql)
            oForm.refreshRow
        END IF
    END SUB
```

Über den Ursprung des Auslösers wird der Button ermittelt: **`oEvent.Source.Model`**. Das Formular ist **`Parent`** zu dem Button. Die Zusatzinformationen werden aus **`oFeld.Tag`** ausgelesen und in einem Array in die beiden Teile aufgeteilt. Im zweiten Teil steht der Name des Feldes, in dem der Primärschlüssel steht. Mit **`Trim(arList(1))`** werden eventuelle Leerzeichen vor und hinter dem Eintrag entfernt.

Ist das Primärschlüsselfeld leer, so wurde der Datensatz noch nicht abgespeichert. Es gibt also auch keine Änderung über den Button zu speichern. Nur wenn in dem Feld ein Wert steht, dann wird ein Kontakt zur Datenbank aufgebaut. Da aber das Formular bereits geöffnet ist müsste dieser Kontakt sowieso bestehen. Die entsprechende Nachfrage ist also nur eine Absicherung.

Schließlich wird der SQL-Code aus den Informationen in den Zusatzinformationen und den zusätzlichen Informationen zu dem Wert in dem Primärschlüsselfeld zusammengestellt. Der SQL-Code wird an die Datenbank weitergegeben und das Ergebnis über **`oForm.refreshRow`** direkt sichtbar im Formular dargestellt.

Der Button kann also so zum einen abspeichern, zum anderen auch direkt den Wert aus der Datenbank wieder auslesen, so dass die Eingabe auf jeden Fall für die Person, die das Formular bedient, erscheint.

Aktuelles Datum oder aktuelle Zeit vollautomatisch über Makros

In dem Formular befinden sich nur zwei beschreibbare Eingabefelder: Das Primärschlüsselfeld und das Namensfeld. Die Werte in den Feldern für das Speicherdatum und das Änderungsdatum bzw. den Speicher-Zeitstempel und den Änderungs-Zeitstempel werden ohne Eingriff des Nutzers automatisch geschrieben. So wird immer die letzte Änderung in dem Änderungsstempel festgehalten.

Makros zum vollautomatischen Einsetzen von aktuellem Datum und aktueller Zeit

Die hier aufgelisteten Makros werden über Formular-**Eigenschaften → Ereignisse → Vor der Datensatzaktion** gestartet.

```
SUB Datum_aktuell
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeldAender AS OBJECT
    DIM unoDate
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByName("datDatum")
    oFeldAender = oForm.getByName("datDatumAender")
```

```
    unoDate = createUnoStruct("com.sun.star.util.Date")
    unoDate.Year = Year(Date)
    unoDate.Month = Month(Date)
    unoDate.Day = Day(Date)
    IF isEmpty(oFeld.Date) THEN
        oFeld.BoundField.updateDate(unoDate)
    END IF
    oFeldAender.BoundField.updateDate(unoDate)
END SUB
```

Der Zugriff auf die Felder im Formular wird erstellt. Anschließend wird das Datum als **`Struct`** zusammengebaut. In dieser Prozedur wird eine Methode gezeigt, wie die einzelnen Werte den verschiedenen Teilen des **`Structs`** für ein Datum zugeordnet werden. Ist das Feld mit dem Namen «datDatum» leer, dann wird das aktuelle Datum über **`oFeld.BoundField.updateDate()`** in das Feld geschrieben. Ansonsten wird das aktuelle Datum nur in das Feld von «datDatumAender» geschrieben. Die endgültige Abspeicherung erfolgt über das Formular. Schließlich wurde das Makro nur direkt vor der Ausführung einer Datensatzaktion eingesetzt, und diese Aktion bedeutet in diesem Falle ein Schreiben oder Ändern, gegebenenfalls auch löschen eines Datensatzes.

```
SUB Datum_Zeit_aktuell
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeldAender AS OBJECT
    DIM unoStmp
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByName("fmtDatumZeit")
    oFeldAender = oForm.getByName("fmtDatumZeitAender")
    unoStmp = createUnoStruct("com.sun.star.util.DateTime")
    WITH unoStmp
        .Year = Year(Date)
        .Month = Month(Date)
        .Day = Day(Date)
        .Hours = Hour(Time)
        .Minutes = Minute(Time)
        .Seconds = Second(Time)
        .NanoSeconds = 000000
    END WITH
    IF isEmpty(oFeld.CurrentValue) THEN
        oFeld.BoundField.updateTimestamp(unoStmp)
    END IF
    oFeldAender.BoundField.updateTimestamp(unoStmp)
END SUB
```

Das Makro für den Zeitstempel läuft vom Prinzip her gleich ab. Hier wird lediglich ein anderes **`Struct`**, nämlich das **`DateTime-Struct`**, befüllt und der entsprechende Wert abgespeichert. Mit **`WITH unoStmp`** beginnt die Zusammensetzung des Structes. **`WITH`** bedeutet hier nur, dass im Prinzip vor jedes der folgenden Elemente unoStmp gesetzt wird, bis eben **`END WITH`** auftaucht. Also **`unoStmp.Year`**, **`unoStmp.Month`** usw.

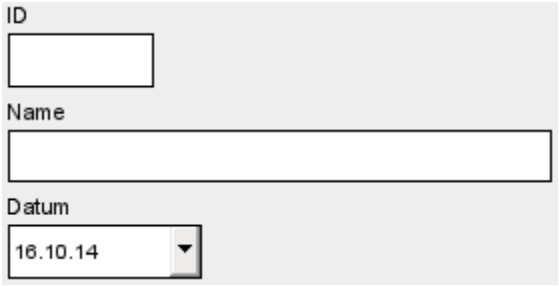
In dem Struct ist es auch möglich, Nanosekunden weiter zu geben. Die spielen hier aber keine Rolle. Das Struct wird also an dieser Position lediglich mit der entsprechenden Anzahl Nullen aufgefüllt.

Schließlich wird der erstelle Timestamp nur dann in das Feld «fmtDatumZeit» geschrieben, wenn das Feld leer ist. Ansonsten überschreibt der Timestamp nur den alten Timestamp im Feld «fmtDatumZeitAender».

Aktuelles Datum oder Zeitstempel über den Standardwert des Formularfeldes

Über die Eigenschaften eines Kontrollfeldes lassen sich Standardwerte festlegen. Beim Datumsfeld ist dies mit **Eigenschaften: Datumsfeld → Allgemein → Standarddatum** zu erreichen. Ent-

sprechend im formatierten Feld über **Eigenschaften: Formatiertes Feld → Allgemein → Standardwert**. Diese Einstellungen sind dann dauerhaft in dem Formular gespeichert. Ein aktuelles Datum oder ein aktueller Zeitstempel können in der grafischen Benutzeroberfläche nicht vorgegeben werden.



Die Standardwerte erscheinen, wie in dem Formular zu sehen, bereits beim Erstellen eines neuen Datensatzes. Sie werden für das Formular allerdings nicht als Datensatzänderung verzeichnet. Erst wenn ein anderes Feld oder eben das Feld mit dem Standardwert geändert wurde nimmt die grafische Benutzeroberfläche das als Änderung wahr und speichert den Datensatz auf die übliche Art und Weise ab (Button »Speichern», Navigation zum nächsten Datensatz, Navigation zum Unterformular u.ä.).

Beim Datumsfeld legt der Standardwert gleichzeitig fest, mit welcher Monatsansicht die Aufklappfunktion des Datumsfeldes zeigt. Werden z.B. häufig Datumswerte eingegeben, die im Vormonat liegen, so kann über ein Standarddatum aus dem Vormonat die Auswahl entsprechend eingestellt werden.

Die Festlegung eines aktuellen Datumswertes bzw. eines aktuellen Zeitstempels als Standardwert erledigen die folgenden Makros. Dabei ist aber zu Bedenken, dass der Wert beim Erreichen eines neuen Datensatzes bereits im Formular erscheint. Es handelt sich also bei der Zeit nicht um den Zeitpunkt, bei dem die Abspeicherung des Datensatzes erfolgte. Bei Datumswerten hat das allerdings weniger Bedeutung, es sei denn, die Bearbeitung eines Datensatzes beginnt z.B. um 23:59 Uhr und endet um 00:01 Uhr des Folgetages.

Makros zur Erstellung des Standardwertes des Datums oder Zeitstempels

Die Makros zur Einstellung von Standardwerten werden in Formular-Eigenschaften → Ereignisse → Nach dem Datensatzwechsel eingetragen. Damit wird das Makro bei jedem neuen Datensatz erneut aufgerufen. Dies ist bei Zeitfeldern besonders wichtig, da bei einer anderen Bindung z.B. die Zeit vom Öffnen des Formulars geschrieben würde.

```
SUB Standarddatum
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM unoDate
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByName("datDatum")
    unoDate = createUnoStruct("com.sun.star.util.Date")
    unoDate.Year = Year(Date)
    unoDate.Month = Month(Date)
    unoDate.Day = Day(Date)
    oFeld.setPropertyValue("DefaultDate", unoDate)
END SUB
```

Das Makro läuft ähnlich ab wie das Makro «Datum_aktuell». Nachdem allerdings das Struct mit dem aktuellen Datum befüllt ist wird jetzt das Standarddatum damit gesetzt. Dies geschieht über den allgemeinen Zugriff auf bestimmte Eigenschaften, nämlich über **`setPropertyValue`**. Die

Eigenschaft heißt beim Datumsfeld «DefaultDate». Der Wert, der hier gespeichert wird, muss vom Typ dem Struct entsprechen.

```
SUB Standarddatum_vorverlegt
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM unoDate
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByName("datDatum")
    unoDate = createUnoStruct("com.sun.star.util.Date")
    i = 2
    IF Month(Date) <= i THEN
        unoDate.Year = Year(Date)-1
        unoDate.Month = 12 - i + Month(Date)
    ELSE
        unoDate.Year = Year(Date)
        unoDate.Month = Month(Date)-i
    END IF
    unoDate.Day = Day(Date)
    oFeld.setPropertyValue("DefaultDate", unoDate)
END SUB
```

Die Vorverlegung des Datums kam als Anfrage in einem Forum. Dort wurden häufig Daten bearbeitet, die die Eingabe eines Datums aus dem Vormonat erforderten. Dann musste in dem aufklappbaren Feld immer erst der Monat zurückgesetzt werden um dann den entsprechenden Datumswert zu setzen.

Wird das Defaultdatum stattdessen direkt auf den Vormonat festgelegt, so kann das Zurückbewegen zum Vormonat entfallen.

Das Makro hat erst einmal den gleichen Aufbau wie «Standarddatum». In der einfachsten Variante müsste jetzt von dem Monatswert lediglich 1 subtrahiert werden, dann würde der Vormonat ausgewählt. Nur schlecht, wenn es sich bei dem Monat um den Januar hat. Dann kommt plötzlich der Monat '0' heraus. Hier müsste also vorher geklärt werden, ob der Monat von der Zahl so groß ist, dass ohne Probleme davon die Korrekturzeit subtrahiert werden kann.

Zuerst wird die Variable **`i`** angegeben, um die die Monatszahl zurückgesetzt werden soll. Hier darf maximal '12' für ein Jahr eingegeben werden. Ist der Monat des aktuellen Datums vom Wert her kleiner oder gleich **`i`**, dann muss das Jahr zurückgesetzt werden: **`Month(Date) <= i`** . Außerdem wird der Monat noch entsprechend angepasst. Der Monat wäre '12', wenn **`i`** und **`Month(Date)`** gleich wären. Ansonsten ist **`i`** größer als **`Month(Date)`** und die Differenz zwischen den beiden Werten wird von 12 subtrahiert, damit auch das Datum entsprechend stimmt.

Nur wenn **`i`** kleiner als **`Month(Date)`** ist wird **`i`** direkt von **`Month(Date)`** subtrahiert und der neue Monat damit festgelegt. Das Jahr bleibt unverändert. Der Tag wird sowieso nicht geändert.

Schließlich wird der berechnet Wert wie beim Makro «Standarddatum» als vorgegebener Standardwert festgelegt.

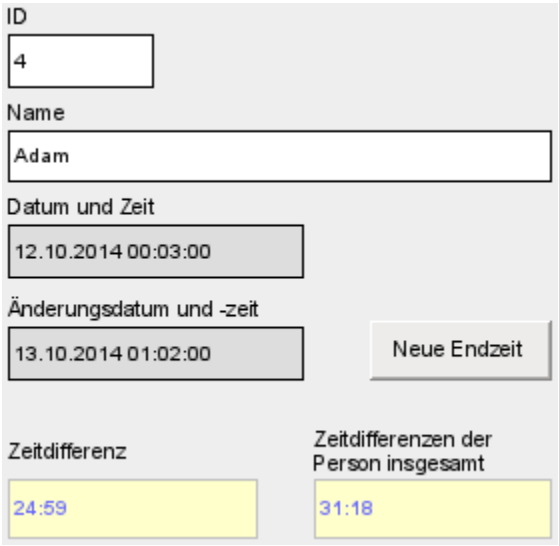
```
SUB StandardZeitstempel
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM stStmp AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByName("fmtZeitstempel")
    stStmp = Year(Date) & "-" & Month(Date) & "-" & Day(Date) & " " & Hour(Time) & ":"
& Minute(Time) & ":" & Second(Time)
```

```
    oFeld.setPropertyValue("EffectiveDefault", stStmp)
END SUB
```

Beim Zeitstempel muss der Wert anders festgelegt werden. Die Formularassistenten bilden häufig den Zeitstempel in zwei getrennten Feldern ab: zum einen ein Datumsfeld, zum anderen ein Zeitfeld. Hier wurde stattdessen ein formatiertes Feld gewählt. Dort ist der differenzierte Aufbau von Datum und Zeit zusammen nicht als Zeitstempelfeld in den Standardwert übertragbar. Stattdessen wird für den Zeitstempel ein Text mit dem entsprechenden Format erstellt. Der Aufbau von stStmp ist hier mit der für Datenbanken üblichen Datumsschreibweise versehen: Zuerst die vierstellige Jahreszahl, gefolgt von '-', dann die zweistellige Monatszahl und wieder '-' und schließlich die zweistellige Tageszahl. Zeit und Datum werden durch eine Leertaste getrennt. Als Verbindungselement der Variablen muss hier das «&» gewählt werden. Wird stattdessen, wie sonst bei Texten gewohnt, ein «+» gewählt, dann werden stattdessen die verschiedenen Werte einfach addiert.

Der Standardwert wird hier über **`EffectiveDefault`** eingefügt. Zu den entsprechenden Standardwerten siehe unter anderen das Base-Handbuch.

Aktuelles Datum oder aktuelle Zeit mit Zeitdifferenzberechnung



Dieses Formular zeigt neben dem Zeitstempel für die erste Abspeicherung und dem Zeitstempel für die Änderung des Datensatzes die Differenz zwischen den Zeiten sowie die Differenz zwischen allen Zeiten, die mit der Person verbunden sind, an. Als Datengrundlage dient die Abfrage,«Zeitdifferenzen_Formularvorlage».

Wird auf den Button «Neue Endzeit» gedrückt, so wird das Änderungsdatum und die Änderungszeit neu gesetzt und die Zeitdifferenzen werden neu eingelesen.

Makros für einen neuen Zeitstempels

```
SUB UpdateTimestamp(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    Datum_Zeit_aktuell
    oFeld = oEvent.Source.Model
    oForm = oFeld.Parent
    IF oForm.IsNew THEN
        oForm.insertRow
    ELSE
        oForm.updateRow
    END IF
    oForm.refreshRow
END SUB
```

Das Makro UpdateTimestamp greift direkt auf das Objekt zu, durch das es ausgelöst wurde: **`oEvent AS OBJECT`**. Dadurch wird im Folgenden das Formular ermittelt, das mit einer (neuen) Endzeit versehen wird. Existiert noch keine Zeiteingabe, so wird die Zeit neu geschrieben. Das Makro «Datum_Zeit_aktuell» wird dafür aus diesem Makro heraus zuerst einmal gestartet.

Mit **`oForm.IsNew`** wird abgesichert, ob ein neuer Datensatz eingefügt (**`insertRow`**) oder ein bestehender Datensatz geändert (**`updateRow`**) werden muss. Anschließend wird der aktuell sichtbare Datensatz neu in das Formular eingelesen: **`oForm.freshRow`**. Damit wird entsprechend auch die Berechnung sichtbar durchgeführt.

Berichte

Für diese Beispieldatenbank wurden zwei Berichte erstellt. Beide greifen auf die gleiche Tabelle «Zeitstempel_Aenderstempel» zu – allerdings über verschiedene Abfragen. Bei dem ersten Bericht ist vor allem mit Formeln innerhalb des Berichtes gearbeitet worden. Beim zweiten Bericht ist das Ergebnis nahezu identisch – nur sind die Inhalte komplett aus der Abfrage übernommen worden, so dass innerhalb des Berichtes keine Rechnung mehr ausgeführt wird.

Zeitdifferenzen in Minuten

Name: *Adam*

StartDatum	Zeitstempel	Zeitstempel Anderung	Differenz in Minuten
11.10.14	11.10.14 09:08	11.10.14 15:27	379
12.10.14	12.10.14 00:03	13.10.14 01:02	1499

Zeitsumme in Minuten: **1878**
Zeitsumme in Stunden: 31:18

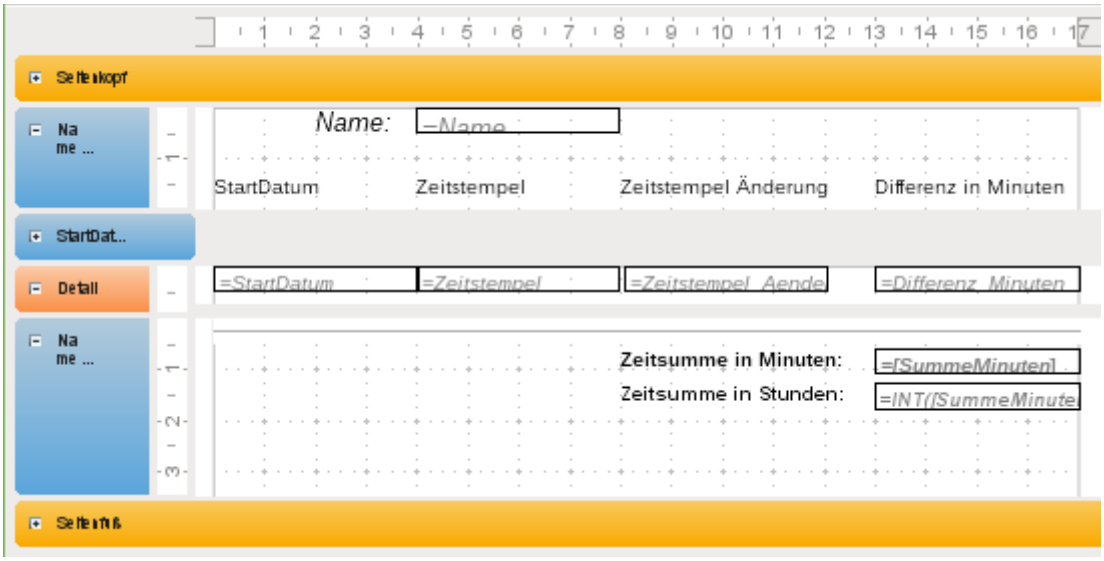
Name: *Eva*

StartDatum	Zeitstempel	Zeitstempel Anderung	Differenz in Minuten
11.10.14	11.10.14 09:12	11.10.14 15:07	355
12.10.14	12.10.14 08:56	12.10.14 15:55	419
13.10.14	13.10.14 08:31	14.10.14 09:30	1499

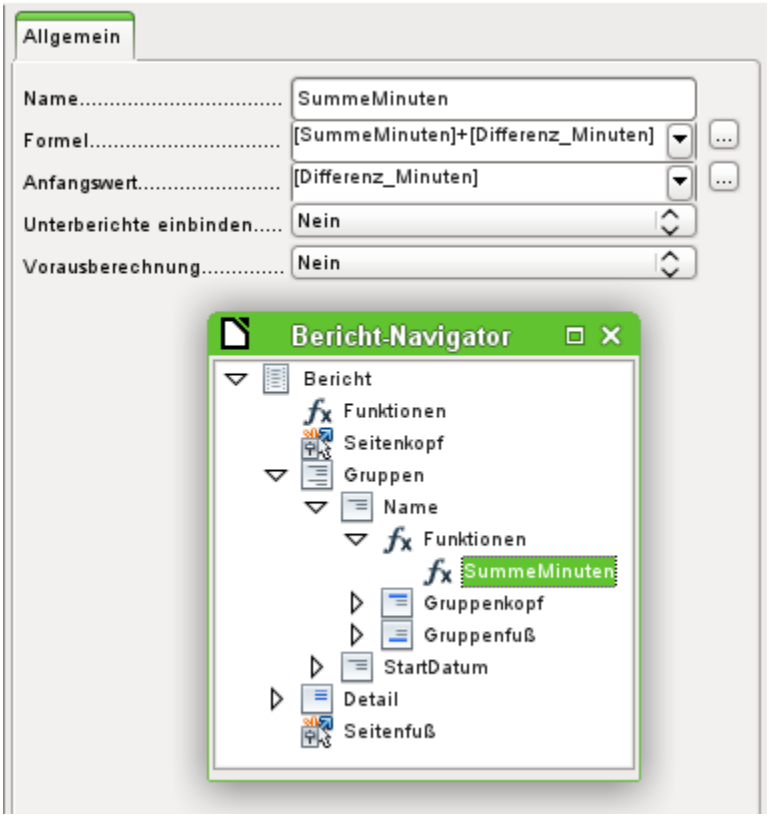
Zeitsumme in Minuten: **2273**
Zeitsumme in Stunden: 37:53

Der Bericht zeigt, aufgelistet nach Personen, die Zeitdifferenzen für jedes einzelne Startdatum in der Maßeinheit Minuten. Außerdem wird die Summe der Zeitdifferenzen in Minuten und schließlich auch in Stunden angegeben. So könnte z.B. eine Zusammenstellung über Arbeitszeiten funktionieren.

Dem Bericht liegt die Abfrage «Zeitdifferenzen» zu Grunde.

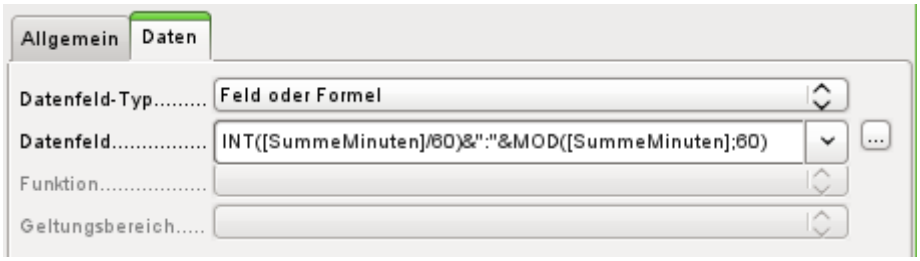


Gruppierungen und Sortierungen greifen beim Report-Builder ineinander. Der Bericht ist zuerst gruppiert nach dem Namen, zu dem es auch einen Gruppenfuß gibt. Dann erfolgt eine weitere Gruppierung nach dem Startdatum. Diese Gruppierung steht zwar in der Übersicht, wird aber im Bericht nicht gezeigt. Dadurch wird lediglich nach dem Startdatum sortiert. Seitenkopf und Seitenfuß sind hier ausgeblendet und stehen in den Eigenschaften auf **Sichtbar → Nein**.



Im Gruppenfuß befindet sich als erstes das Feld für die Summierung der Summe. Die Funktion wurde händisch erstellt, da die Automatik von LO leider versagte. Über den Berichtsnavigator ist so eine Funktion anschließend erreichbar und kann dann auch noch bearbeitet werden.

`[Differenz_Minuten]` ist der Feldwert aus der Abfrage «Zeitdifferenzen». Der erste Wert wird gelesen, als Wert der Formel gespeichert und anschließend wird der nächse Wert der Abfrage einfach hinzu addiert. Auf die Formel kann anschließend wie auf das Feld der Abfrage mit eckigen Klammern zugegriffen werden: **`[SummeMinuten]`**.



Für die Zeitsumme in Stunden wird bei den Daten des Feldes eine Formel eingetragen. Sie liest zuerst den Wert der Funktion **`[SummeMinuten]`** aus und teilt diesen durch 60. Mit der Funktion **`INT`** werden daraus nur die Ganzzahlbestandteile übernommen. Anschließend wird mit **`&":"&`** die Anzahl der verbleibenden Minuten angehängt. **`MOD([SummeMinuten];60)`** stellt den «Rest» der Division der Minutenanzahl durch 60 dar.

Zeitdifferenzen berechnet in einer Abfrage in Stunden

Name: *Adam*

StartDatum	Zeitstempel	Zeitstempel Anderung	Differenz in Stunden
11.10.14	11.10.14 09:08	11.10.14 15:27	6:19
12.10.14	12.10.14 00:03	13.10.14 01:02	24:59
		Zeitsumme in Stunden	31:18

Name: *Eva*

StartDatum	Zeitstempel	Zeitstempel Anderung	Differenz in Stunden
11.10.14	11.10.14 09:12	11.10.14 15:07	5:55
12.10.14	12.10.14 08:56	12.10.14 15:55	6:59
13.10.14	13.10.14 08:31	14.10.14 09:30	24:59
		Zeitsumme in Stunden	37:53

Dieser Bericht ist äußerlich gleich dem vorhergehenden aufgebaut. Nur ist hier die Differenz zwischen den Zeiten direkt in Stunden ausgedrückt.

Dem Bericht liegt statt der Abfrage «Zeitdifferenzen» die Abfrage «Zeitdifferenzen_Formularvorlage» zu Grunde. In dieser Abfrage werden bereits die Zeiten berechnet, auch die Gesamtzeiten gruppiert nach dem Namen. Dadurch ist für den Bericht keine Funktion und keine Formel notwendig.

Fortlaufende gruppierte Summierungen erstellen

Wo ist im Haushalt bloß das Geld geblieben. Das Nachverfolgen des Finanzstromes lässt sich mit entsprechenden Abfragetechniken direkt innerhalb einer Abfrage auch berechnen und dann noch zusätzlich in Formularen übersichtlich geordnet nach Konten und Kategorien darstellen.

Die Datenbank «Beispiel_Saldo_fortlaufend» verzichtet dabei nahezu komplett auf Makros. Lediglich ein kleines Makro zur Aktualisierung eines Formulars erleichtert etwas die Übersicht.

Tabellen

In der Tabelle "Kasse" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.

Feldname	*Feldtyp*	*Beschreibung*
Betrag	Zahl	Der Betrag, der aus dem folgenden Konto abgebucht oder hinzu gebucht wird. Bei der Felderstellung ist auf 2 Nachkommastellen zu achten. **Feldeigenschaften → Eingabe erforderlich → Ja**
Konto_ID	Tiny Integer	Fremdschlüssel aus der Tabelle "Konto". **Feldeigenschaften → Eingabe erforderlich → Ja**
Datum	Datum	Hier soll das Datum gespeichert werden, zu dem die Einnahme/Ausgabe erfolgt ist. **Feldeigenschaften → Eingabe erforderlich → Ja**
Kategorie_ID	Tiny Integer	Fremdschlüssel aus der Tabelle "Kategorie". Wird keine Kategorie zugewiesen, so wird der Betrag ohne Kategorie geführt.
Adressat_ID	Integer	Fremdschlüssel aus der Tabelle "Adressat".
Verwendungszweck	Text	Wofür ist das Geld verwandt worden? Dieses Feld speichert Zusatzinformationen zu den Kategorien.
Umbuch_Konto_ID	Tiny Integer	Wird Geld z.B. vom Girokonto abgehoben, so wechselt das Geld lediglich das Konto von «Giro» zu «Bargeld», ist aber noch nicht anderweitig ausgegeben.

Um die Umbuchung von einem Konto zum anderen Konto so abzusichern, dass tatsächlich von einem Konto zum anderen gebucht wird und nicht das abgebende Konto gleich dem aufgebenden Konto ist, wird innerhalb der Tabelle ein **`CONSTRAINT`** definiert. Über **Extras → SQL** wird eingegeben:

```
ALTER TABLE "Kasse" ADD CONSTRAINT "Konto ungleich Umbuchungskonto"
CHECK ("Umbuch_Konto_ID" <> "Konto_ID");
```

Sofern ein Eintrag im Feld "Umbuch_Konto_ID" erstellt wird wird dieser Vergleich vorgenommen. Das Feld "Umbuch_Konto_ID" darf also sehr wohl leer sein, darf eben nur nicht den gleichen Wert wie "Konto_ID" haben.

Die Tabelle "Adressat" kann den Einstieg für eine komplette Adressverwaltung liefern. Hier soll erst einmal nur eine Bezeichnung für die Person oder Firma stehen. Würde ein komplettes Buchungsprogramm erstellt, so würde einfach die Tabelle um die entsprechenden Adressfelder und genauere Daten wie Vorname und Anrede erweitert.

In der Tabelle "Adressat" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Adressat	Text	Woher stammt die Rechnung, wer erhält das Geld? **Feldeigenschaften → Eingabe erforderlich → Ja**

Es soll festgestellt werden, wofür das Geld ausgegeben wurde. Als Beispielkategorien sind hier 'Einkünfte', 'Wohnen', 'Ernährung' usw. vorgegeben.

In der Tabelle "Kategorie" werden folgende Felder definiert:

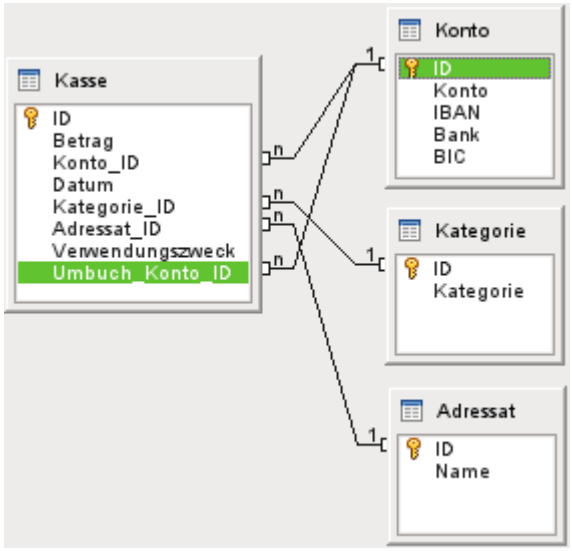
Feldname	*Feldtyp*	*Beschreibung*
ID	Tiny Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Die Kategorien bleiben überschaubar, so dass lediglich eine kleine Zahl ohne Auto-Wert notwendig ist.

Feldname	*Feldtyp*	*Beschreibung*
Kategorie	Text	Wofür ist die Ausgabe erfolgt oder woher kommt die Einnahme? **Feldeigenschaften → Eingabe erforderlich → Ja**

In der Tabelle "Konto" werden folgende Felder definiert:

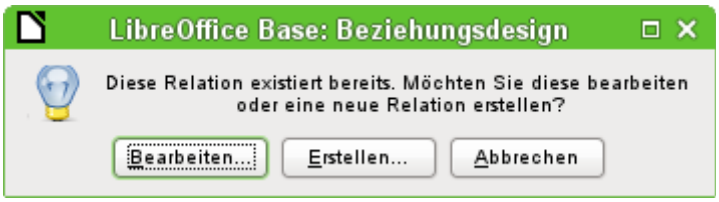
Feldname	*Feldtyp*	*Beschreibung*
ID	Tiny Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Die Kontoverbindungen bleiben überschaubar, so dass lediglich eine kleine Zahl ohne Auto-Wert notwendig ist.
Konto	Text	Bezeichnung für das Konto, wie es auch in den Listenfeldern erscheint. **Feldeigenschaften → Eingabe erforderlich → Ja**
IBAN	Text	IBAN-Kontobezeichnung
Bank	Text	Bezeichnung der Bankverbindung
BIC	Text	BIC-Bankbezeichnung für internationalen Zahlungsverkehr

Über **Extras → Beziehungen** werden die Tabellen miteinander verbunden:

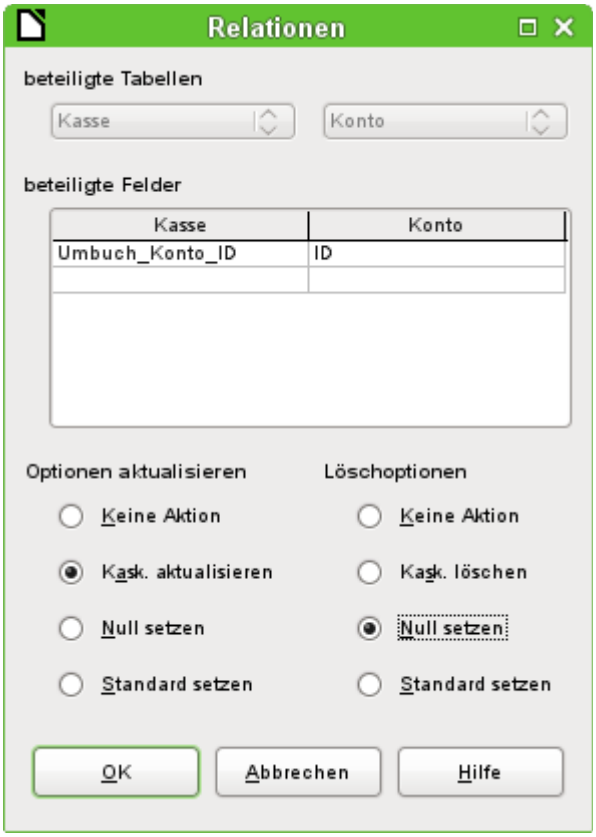


Die Tabelle "Konto" ist dabei sowohl mit dem Feld "Kasse"."Konto_ID" als auch mit dem Feld "Kasse"."Umbuch_Konto_ID" verbunden. Für beide Felder werden die gleichlautenden Kontobezeichnungen genutzt. Über die vorher definierte Bedingung ist ausgeschlossen, dass beide Felder den gleichen Wert haben dürfen.

Beim Erstellen der zweiten Verbindung erscheint die folgende Nachfrage:



Die bereits existierende Beziehung zwischen "Konto"."ID" und "Kasse"."Konto_ID" soll nicht bearbeitet werden. Stattdessen wird eine weitere Beziehung erstellt. Es erscheint der folgende Dialog zu den Relationen:

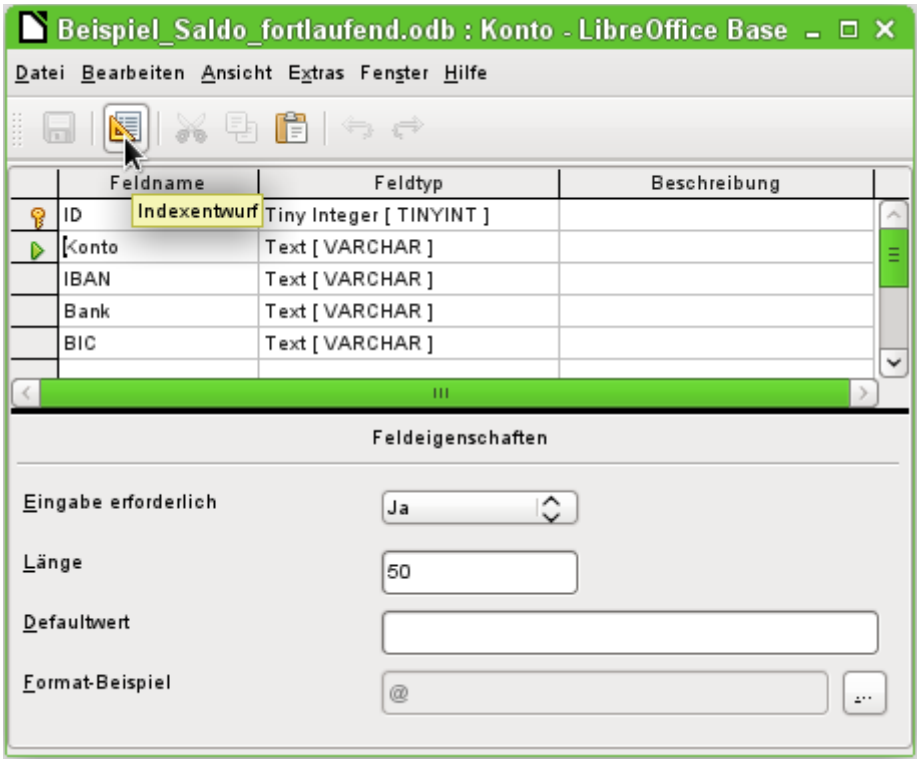


Wird die "Konto."ID" aktualisiert, so wird auch der Eintrag in "Kasse"."Umbuch_Konto_ID" angepasst. Wird die "Konto"."ID" gelöscht, so wird der Eintrag in "Kasse"."Umbuch_Konto_ID" auf **NULL** gesetzt. Die Umbuchung wird also zurückgenommen, da das Konto aufgelöst ist. Der Wert des Kontos muss an anderer Stelle gut geschrieben werden.

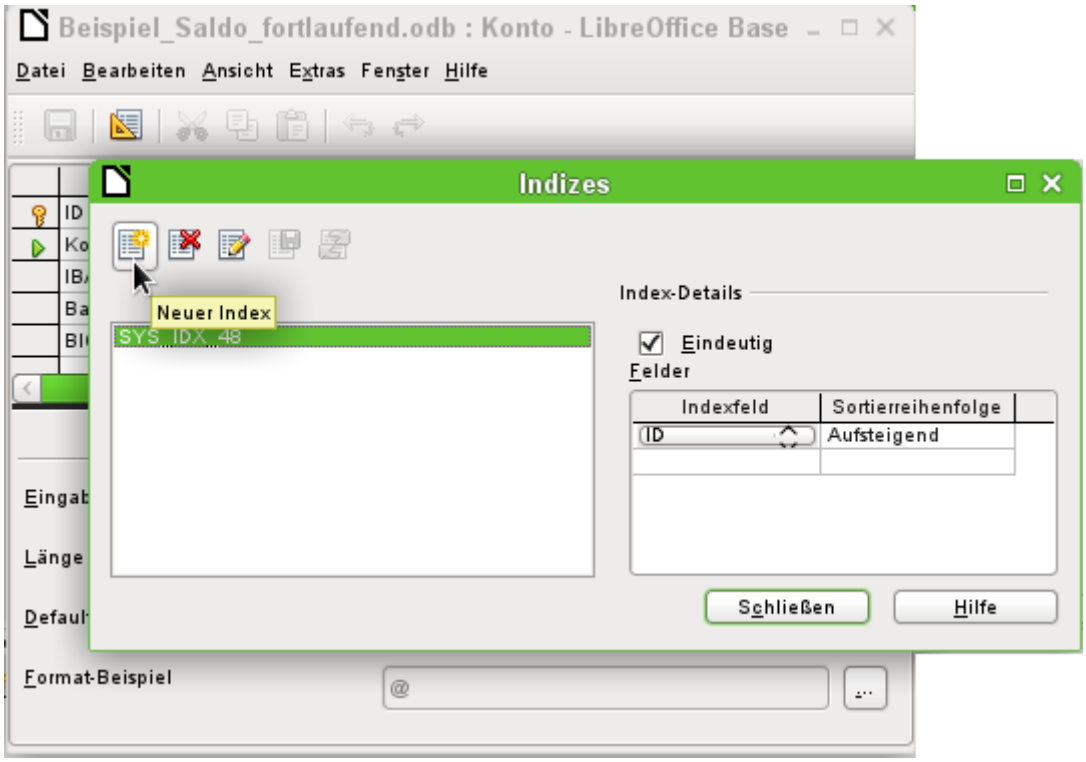
Da aber noch eine zweite Verbindung zur "Konto"."ID" existiert, die nie **NULL** werden kann, ist eine Löschung des Kontos sowieso nicht möglich.

Die anderen Verbindungen der Tabelle "Kategorie" zur Tabelle "Kasse" und der Tabelle "Adresse" zur Tabelle "Kasse" werden entsprechend der obigen Abbildung gesetzt: **Kaskadiert aktualisieren** und beim Löschen **Null setzen**.

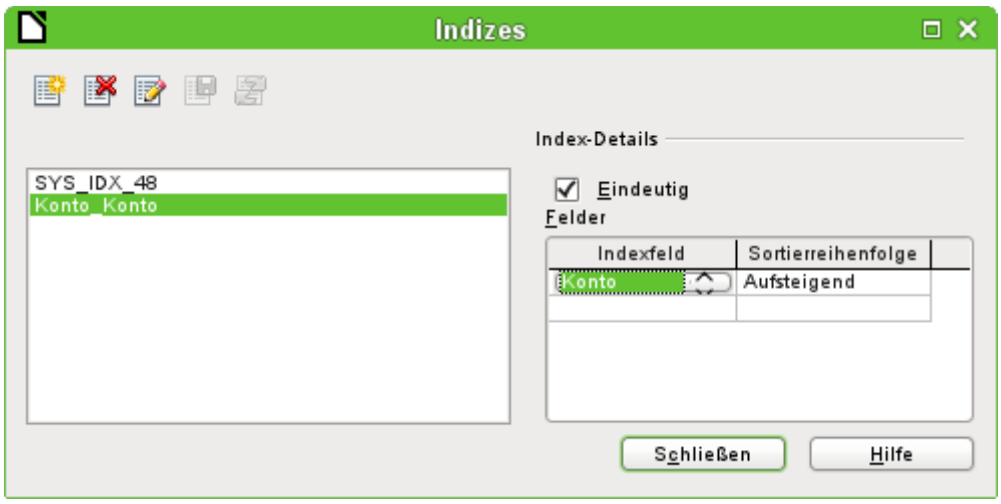
Anschließend werden noch den Feldern "Konto"."Konto", "Kategorie"."Kategorie" und "Adressat"."Name" ein eindeutiger Index zugewiesen, damit diese Felder anschließend auch für Listenfelder in Formularen sinnvoll genutzt werden können.



Im Tabellenentwurf wird der Indexentwurf gestartet.



Es besteht bereits ein Index für das Primärschlüsselfeld "ID". Für das Feld "Konto" wird in der Tabelle "Konto" ein neuer Index erstellt.



Der Index für das Feld "Konto" muss auf jeden Fall auch eindeutig sein: **Index-Details → Eindeutig → Ja**. Die Sortierreihenfolge ist sinnvollerweise aufsteigend. Diese Reihenfolge wird auch bei der Erstellung von Abfragen automatisch berücksichtigt, ist allerdings nachrangig gegenüber dem Primärschlüssel.

Standardmäßig wird der Index als «Index1» bezeichnet. Hier sollte auf jeden Fall nachgebessert werden. Im obigen Bild drückt der Name bereits aus, dass der Indes für die Tabelle "Konto" und das darin enthaltene Feld "Konto" steht. Datenbankweit müssen die Indizes eindeutig sein. Base schafft es nicht, die Indizes so zu verwalten, dass beim nächsten Index einfach hochgezählt wird, so dass auch dort «Index1» vorgeschlagen würde. Dies führt dann zwangsläufig zu einer Fehlermeldung.

Zu Bestätigung des Indexes ist jetzt auf jeden Fall noch das vierte Symbol von links zu wählen.[2] Erst dann wird der Index auch gespeichert und der Dialog kann geschlossen werden.

Sinnvollerweise sollten die Felder, denen jetzt ein Index zugeteilt wurde, natürlich auch immer Daten besitzen. Es sollte auf jeden Fall unter **Eingabe Erforderlich → Ja** gewählt werden.

Abfragen

Die Abfragen sind nach der Komplexität gestaffelt. Sie sollen die verschiedenen Zugriffsweisen auf die Berechnungen eines fortlaufenden Saldos aufzeigen. Dabei wird grundsätzlich auf korrelierende Unterabfragen zugegriffen, da nur mit dieser Abfrageart gleichzeitig alle Daten dargestellt werden können und ein entsprechender Verlauf aufgezeigt werden kann. Außerdem hat diese Abfragetechnik den Vorteil, dass sie sich gut für bearbeitbare Abfrage nutzen lässt, so dass letztlich schon ein einfaches Tabellenkontrollfeld in einem Formular zur Eingabe von Daten und zur Übersicht ausreicht.

Saldo

Die Abfrage «Saldo» soll lediglich zeigen, dass es nicht notwendig ist, bei Beträgen mit zwei Spalten zu arbeiten, wie dies in der Buchführung mit «Soll» und «Haben» üblich ist.

Einnahmen und Ausgaben lassen sich aufgrund der Vorzeichen leicht auf zwei Spalten verteilen. Zusätzlich wird noch das fortlaufende Saldo dargestellt, hier einfach fortlaufend nach dem Primärschlüssel.

Diese Abfrage wird nachher in den Formularen nicht weiter genutzt sondern soll nur das Prinzip aufzeigen.

2 Das Symbol zum Speichern beinhaltet weiterhin eine Diskette. Mittlerweile gibt es sicher viele Nutzer, denen sich der Sinn dieses Symbols mangels entsprechendem Datenspeicher erst mit Erklärungen erschließt.

	ID	Einnahme	Ausgabe	Saldo
▶	1	158	0	158
	3	0	-12,35	145,65
	4	0	-215	-69,35
	5	16,63	0	-52,72
	6	25,36	0	-27,36
	7	3	0	-24,36
	8	12	0	-12,36
	9	0	-7	-19,36
	10	3,45	0	-15,91
	11	13,5	0	-2,41
	12	13,5	0	11,09
☼	<Autol			

```
SELECT "ID", CASEWHEN( "Betrag" > 0, "Betrag", 0 ) AS "Einnahme",
   CASEWHEN( "Betrag" < 0, "Betrag", 0 ) AS "Ausgabe",
   ( SELECT SUM( "Betrag" ) FROM "Kasse"
      WHERE "ID" <= "a"."ID" AND "Umbuch_Konto_ID" IS NULL )
      AS "Saldo"
FROM "Kasse" AS "a"
WHERE "Umbuch_Konto_ID" IS NULL
ORDER BY "ID" ASC
```

Mit **CASEWHEN** wird abgefragt, ob der angegebene Betrag größer als 0 oder kleiner als 0 ist. Für alle anderen Fälle wird stattdessen eine 0 in die Felder für die Ausgabe geschrieben. Ist der Betrag größer als 0, so wird der Alias "Einnahme" gewählt. Ist der Betrag kleiner als 0, so werden die Werte dem Alias "Ausgabe" zugeordnet.

Die Spalte "Saldo" wird durch eine korrelierende Unterabfrage erstellt. Es wird die Summe der Einträge im Feld "Betrag" aus der Tabelle "Kasse" erstellt. Diese Summe wird für alle Felder erstellt, deren Primärschlüssel "ID" kleiner oder gleich dem Primärschlüssel des aktuellen Datensatzes **"a"."ID"** ist. Es werden nur die Felder summiert, bei denen im Feld "Umbuch_Konto_ID" der Tabelle "Kasse" kein Eintrag erfolgt ist. Bei Umbuchungen wird schließlich das Geld nur von einem Konto zum anderen Konto transferiert. Der Wert des einen Kontos fällt um den gleichen Betrag wie der Wert des anderen Kontos steigt. Im Saldo aller Konten ändert sich nichts.

Die Tabelle der äußeren Abfrage ist mit einem Alias versehen: **"Kasse" AS "a"**. Dies ist erforderlich, damit die korrelierende Unterabfrage auf den aktuellen Datensatz zugreifen kann. Auch die äußere Abfrage zeigt nur die Datensätze, bei denen keine Umbuchung erfolgt ist. Aus diesem Grunde fehlen die Datensätze für die Primärschlüssel "ID" = 2 und 13, wie sie in den folgende Abfragen schließlich doch noch auftauchen.

Sortiert wird die äußere Abfrage nach dem Primärschlüssel, da nach dem Primärschlüssel auch der Saldo ermittelt wurde.

Ansicht_Kasse_mit_Umbuchungen

Soll aus mehreren Abfragen auf einen Datenbestand zugegriffen werden, der über eine Abfrage erstellt wurde, so empfiehlt sich dafür eine Ansicht. Ansichten werden häufig zuerst als Abfrage erstellt. Anschließend mit einem rechten Mausklick auf die Abfrage «Als Ansicht erstellen» gewählt.

Taucht diese Möglichkeit nicht im Kontextmenü auf, so bedeutet dies nicht, dass sie nicht verfügbar ist. Das Kontextmenü scheint hier etwas eigenwillig zu sein. Hier hilft dann: Abfrage markieren und über **Bearbeiten** → **Ansicht** erstellen den Weg zu einer Ansicht frei zu machen.

ID	Betrag	Konto_ID	Konto	Datum	Kategorie_ID	Kategorie	Adressat_ID	Name	Verwendun
1	158,00	2	Sparbuch	01.01.12	1	Einkünfte			
2	-37,00	1	Giro	01.01.12	1	Einkünfte			
2	37,00	3	Bargeld	01.01.12	1	Einkünfte			
3	-12,35	1	Giro	02.01.12	1	Einkünfte			
4	-215,00	2	Sparbuch	03.01.12	2	Wohnen			
5	16,63	1	Giro	03.01.12	3	Ernährung			
6	25,36	1	Giro	03.01.12	4	Mobilität			
7	3,00	2	Sparbuch	04.01.12	5	Gesundheit			
8	12,00	1	Giro	05.01.12	3	Ernährung			
9	-7,00	1	Giro	07.01.12	3	Ernährung			
10	3,45	1	Giro	09.01.12	2	Wohnen			
11	13,50	2	Sparbuch	12.01.12	3	Ernährung			
12	13,50	1	Giro	10.01.12	1	Einkünfte			
13	-215,00	1	Giro	03.01.12	2	Wohnen			
13	215,00	3	Bargeld	03.01.12	2	Wohnen			

Die Ansicht erstellt eine lesbare Übersicht über den Inhalt der Tabelle "Kasse". Lesbar in sofern, als zusätzlich zu den Fremdschlüsseln hier die entsprechenden Begriffe aus den verbundenen Tabellen angezeigt werden.

Außerdem führt die Ansicht die Datensätze für den Primärschlüssel "ID" doppelt auf, wenn es sich um Umbuchungen handelt. Aus diesem Gründe sind für "ID" = 2 und 13 jeweils zwei Datensätze zu sehen. Sowohl das Konto, von dem aus gebucht wird, als auch das Konto, zu dem hin gebucht wird, werden so repräsentiert. Eine zusätzliche Eingabe ist hierfür also nicht erforderlich.

```
SELECT "Kasse"."ID", "Kasse"."Betrag", "Kasse"."Konto_ID",
"Konto"."Konto", "Kasse"."Datum", "Kasse"."Kategorie_ID",
"Kategorie"."Kategorie", "Kasse"."Adressat_ID", "Adressat"."Name",
"Kasse"."Verwendungszweck"
FROM "Kasse"
LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
LEFT JOIN "Kategorie" ON "Kasse"."Kategorie_ID" = "Kategorie"."ID"
LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
UNION
SELECT "Kasse"."ID", "Kasse"."Betrag" * -1 AS "Betrag",
"Kasse"."Umbuch_Konto_ID", "Konto"."Konto", "Kasse"."Datum",
"Kasse"."Kategorie_ID", "Kategorie"."Kategorie",
"Kasse"."Adressat_ID", "Adressat"."Name", "Kasse"."Verwendungszweck"
FROM "Kasse"
LEFT JOIN "Konto" ON "Kasse"."Umbuch_Konto_ID" = "Konto"."ID"
LEFT JOIN "Kategorie" ON "Kasse"."Kategorie_ID" = "Kategorie"."ID"
LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
WHERE NOT "Kasse"."Umbuch_Konto_ID" IS NULL
```

Die Ansicht verbindet zwei Abfragen mit dem Befehl **`UNION`**. Der Befehl **`UNION`** sorgt dafür, dass gleichlautende Datensätze aus der zweiten Abfrage nicht erscheinen. Außerdem sortiert der Befehl automatisch nach dem ersten Feld der Abfrage. Die Feldreihenfolge beider Abfragen muss gleich sein. Auf jeden Fall müssen die Felder vom Format her zueinander passen.

Zuerst werden alle Felder zusammen mit der dazugehörigen Tabellenbezeichnung aufgelistet. Anschließend wird die Tabelle "Kasse" als maßgebende Tabelle für alle Datensätze erklärt. Über **`LEFT JOIN`** werden alle anderen Tabellen hiermit verbunden. Die links stehende Tabelle zeigt also alle Datensätze aus ihrem Inhalt auf jeden Fall auf, auch wenn vielleicht eine andere Tabelle in dem Verbund nicht für alle Datensätze liefert. Würde hier nicht mit einem **`LEFT JOIN`** gearbeitet, so bliebe die Abfrage leer, da die Tabelle "Adressat" in dem Beispiel noch komplett ohne Daten ist.

In der zweiten Abfrage werden die gleichen Felder deklariert. Nur wird hier der Inhalt des Feldes "Betrag" mit '-1' multipliziert. Schließlich soll hier die Gegenbuchung erfolgen. Es ist also auch möglich, bei der ersten Buchung einen positiven Betrag einzugeben und dennoch eine Umbuchung mit einem Umbuchkonto zu wählen. Dann wird einfach ein negativer Betrag von dem Umbuchkonto gebucht.

In der zweiten Abfrage sollen nur die Datensätze berücksichtigt werden, bei denen die "Umbuch_Konto_ID" aus der Tabelle "Kasse" gleich der "ID" aus der Tabelle "Konto" ist. Außerdem sollen nur die Felder auftauchen, die eben in der Tabelle "Kasse" tatsächlich im Feld "Umbuch_Konto_ID" einen Eintrag haben: **`WHERE NOT "Kasse"."Umbuch_Konto_ID" IS NULL`**.

Durch die Bedingung wird ausgeschlossen, dass für lauter leere Kontoeinträge noch zusätzlich ein Datensatz angezeigt wird.

Sind die Einträge in "Konto"."Konto", "Kategorie"."Kategorie" und "Adressat"."Name" nicht eindeutig, so müsste noch zusätzlich das Feld mit dem jeweiligen Primärschlüssel mitgeführt werden.

Kontoverlauf

Mit dieser Abfrage sollen die Buchungsbeträge und der Kontostand für die jeweiligen Konten in Abhängigkeit zum Datum aufgelistet werden. Diese Abfrage kann auch gut für die spätere Erstellung eines Berichtes genutzt werden.

Konto	Datum	Betrag	Kontoverlauf
Bargeld	01.01.12	37,00	37
Bargeld	03.01.12	215,00	252
Giro	01.01.12	-37,00	-37
Giro	02.01.12	-12,35	-49,35
Giro	03.01.12	16,63	-32,72
Giro	03.01.12	25,36	-7,36
Giro	03.01.12	-215,00	-222,36
Giro	05.01.12	12,00	-210,36
Giro	07.01.12	-7,00	-217,36
Giro	09.01.12	3,45	-213,91
Giro	10.01.12	13,50	-200,41
Sparbuch	01.01.12	158,00	158
Sparbuch	03.01.12	-215,00	-57
Sparbuch	04.01.12	3,00	-54
Sparbuch	12.01.12	13,50	-40,5

```
SELECT "a"."Konto", "a"."Datum", "a"."Betrag",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen"
      WHERE "Konto" = "a"."Konto" AND ( "Datum" < "a"."Datum"
      OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Kontoverlauf"
FROM "Ansicht_Kasse_mit_Umbuchungen" AS "a"
ORDER BY "a"."Konto" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Wieder handelt es sich um eine korrelierende Unterabfrage, die dazu dient, den entsprechenden Kontoverlauf auf zu listen. Als Grundlage für die gesamte Abfrage dient die vorher erstellte Ansicht. Da die Ansicht grundsätzlich nicht editierbar ist, ist eine allein darauf zurückgreifende Abfrage natürlich auch nicht editierbar.

In der Unterabfrage wird die Summe des Betrages unter mehreren Bedingungen ermittelt:

1. Das Konto muss gleich dem Konto der äußeren Abfrage sein.
2. Das Datum muss kleiner dem Datum der äußeren Abfrage sein oder
3. das Datum ist gleich dem Datum der äußeren Abfrage und das Schlüsselfeld "ID" ist kleiner oder gleich dem Schlüsselfeld der äußeren Abfrage.

Bei dieser Bedingung ist auf die korrekte Klammerung zu achten. Die Datumsbedingungen sind gemeinsam in eine Klammer gesetzt worden, da sonst die Kontobedingung und die erste Datumsbedingung abgearbeitet würden oder die zweite Datumsbedingung für alle Konten, nicht nur für das des aktuellen Datensatzes.

Das Schlüsselfeld wird in der zweiten Bedingung herangezogen, da dies auf jeden Fall unterschiedlich bei gleichem Datum sein wird.

Die gesamte Abfrage wird nach dem Konto, anschließend dem Datum, und zur eindeutigen Sortierung dem Primärschlüsselfeld ID sortiert.

Kategorieverlauf

Der Kategorieverlauf entspricht vom Aufbau her dem Kontoverlauf. Aus dieser Abfrage resultiert später wieder der Inhalt eines Berichtes und gegebenenfalls auch Informationen in Formularen.

```
SELECT "a"."Kategorie", "a"."Datum", "a"."Betrag",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen"
      WHERE "Kategorie" = "a"."Kategorie" AND
      ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Kategorieverlauf"
FROM "Ansicht_Kasse_mit_Umbuchungen" AS "a"
ORDER BY "a"."Kategorie" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Anmerkungen zum Code siehe *Kontoverlauf*.

Saldo_Konto

Diese Abfrage wird benötigt, um den aktuellen Saldo eines Kontos innerhalb eines Formulars dar zu stellen. Sie muss nicht editierbar sein und kann sich daher direkt auf die «Ansicht_Kasse_mit_Umbuchungen» beziehen.

```
SELECT "Konto_ID", "Konto" AS "Kontoname",
   SUM( "Betrag" ) AS "Saldo_Konto",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen"
      WHERE "Datum" <= :qDatum ) AS "Saldo_gesamt"
FROM "Ansicht_Kasse_mit_Umbuchungen"
WHERE "Datum" <= :qDatum
GROUP BY "Konto_ID", "Konto"
ORDER BY "Kontoname"
```

Bis auf die eingebaute Unterabfrage lässt sich der Code direkt über die GUI zusammenstellen. Quelle ist "Ansicht_Kasse_mit_Umbuchungen". Aus der Quelle werden bloß Datensätze bis zu einem bestimmten Datum ausgelesen, das über eine Parameterabfrage bezogen wird. Die erstellte Summe des Betrages wird nach der "Konto_ID" und dem "Konto" gruppiert. Dadurch wird für jedes Konto eine separate Summierung erstellt.

Schließlich wird in einer Unterabfrage noch eine Summe über alle Beträge unabhängig von dem Konto erstellt. Auch diese Unterabfrage zeigt das Ergebnis nur bis zum dem Datum, das über die Parameterabfrage übergeben wird.

Saldo_Kategorie

Diese Abfrage erledigt vom Prinzip her das, was die Abfrage «Saldo_Konto» für das Konto erledigt, dieses Mal für die Kategorie. Auch das Saldo der Kategorie muss nicht editierbar sein und lässt sich aus der «Ansicht_Kasse_mit_Umbuchungen» ermitteln.

```
SELECT "Kategorie_ID", "Kategorie",
   SUM( "Betrag" ) AS "Saldo_Kategorie",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen"
      WHERE "Datum" <= :qDatum ) AS "Saldo_gesamt"
FROM "Ansicht_Kasse_mit_Umbuchungen"
WHERE "Datum" <= :qDatum
GROUP BY "Kategorie_ID", "Kategorie"
ORDER BY "Kategorie"
```

Der Gesamtsaldo der Abfrage «Saldo_Kategorie» ist gleich dem Gesamtsaldo der Abfrage «Saldo_Konto».

Datum_Max

```
SELECT MAX( "Datum" ) AS "Datum_Max" FROM "Kasse"
```

Diese Abfrage wird lediglich dazu genutzt, innerhalb eines Formulars andere Abfragen einzubinden und dabei den jeweiligen Endstand des Kontos zu ermitteln. Sie gibt das jüngste Datum **`(MAX ("Datum"))`** an die Parameterabfragen «Saldo_Kategorie» und «Saldo_Konto» weiter.

monatlich_Konto

Die Abfragen «monatlich_Konto» und «monatlich_Kategorie» dienen dazu, eine monatliche Übersicht über die Finanzbewegungen in Form von Diagrammen zu erstellen. Beide Abfragen werden in den weiter unten aufgezeigten Berichten verwandt.

```
SELECT YEAR( "Datum" ) || '-' || RIGHT( '0' || MONTH( "Datum" ), 2 )
   AS "JahrMonat", "Konto", SUM( "Betrag" ) AS "Summe"
FROM "Ansicht_Kasse_mit_Umbuchungen"
GROUP BY YEAR( "Datum" ), MONTH( "Datum" ), "Konto"
ORDER BY "JahrMonat" ASC
```

Als Zeiteinteilung wird zuerst das Jahr und dann die Monatszahl zusammen gefasst. Um später eine entsprechende Sortierung im Diagramm erhalten zu können muss die Monatszahl so dargestellt werden, dass bei einstelligen Monaten eine führende Null erzeugt wird. Mit **`'0' || MONTH( "Datum" )`** wird vor die Monatszahl eine '0' geschrieben, unabhängig davon, ob die Zahl einstellig oder zweistellig ist. Mit **`RIGHT( … , 2 )`** werden dann die beiden am weitesten rechts liegenden Zeichen weiter genutzt.

Datenquelle für die Abfrage ist die vorher erstellte Ansicht. In dieser steht bereits ausgeschrieben der Name des Kontos. Die Beträge werden aufsummiert. Da die gesamte Abfrage nach Jahr, Monat und Konto gruppiert wird erscheinen die Beträge aufsummiert nach genau diesen Kriterien, also für jedes Konto und jeden Monat und jedes Jahr entsprechend aufgetrennt.

monatlich_Kategorie

```
SELECT YEAR( "Datum" ) || '-' || RIGHT( '0' || MONTH( "Datum" ), 2 )
   AS "JahrMonat", "Kategorie", SUM( "Betrag" ) AS "Summe"
FROM "Ansicht_Kasse_mit_Umbuchungen"
GROUP BY YEAR( "Datum" ), MONTH( "Datum" ), "Kategorie"
ORDER BY "JahrMonat" ASC
```

Das Vorgehen bei der Abfrage «monatlich_Kategorie» ist identisch dem Vorgehen bei der Abfrage «monatlich_Konto».

Konto_Saldo_Konto_Kategorie_Adressat

Nahezu alle vorhergehenden Abfrageergebnisse zusammenfassen in einer Abfrage – auch das geht. Die folgende Abfrage berücksichtigt dabei nicht das Feld für die Umbuchungen, ist also die etwas einfacher gehaltene Version. Kennzeichen beider Abfragen ist aber, dass sie trotz der Fülle an Informationen weiter editierbar bleiben, also direkt in Formularen genutzt werden können.

ID	Betrag	Konto_ID	Datum	Kategorie_ID	Adressat_ID	Verwendungszweck	Umbuch_Konto_ID
1	158,00	2	01.01.12	1			
2	-37,00	1	01.01.12	1			3
3	-12,35	1	02.01.12	1			
4	-215,00	2	03.01.12	2			
5	16,63	1	03.01.12	3			
6	25,36	1	03.01.12	4			
13	-215,00	1	03.01.12	2			3
7	3,00	2	04.01.12	5			
8	12,00	1	05.01.12	3			
9	-7,00	1	07.01.12	3			
10	3,45	1	09.01.12	2			
12	13,50	1	10.01.12	1			
11	13,50	2	12.01.12	3			
<Autol							

Einnahme	Ausgabe	Kontoname	Saldo_Konto	Kategoriename	Saldo_Kategorie	Saldo_gesamt
158	0	Sparbuch	158	Einkünfte	158	158
0	-37	Giro	-37	Einkünfte	121	121
0	-12,35	Giro	-49,35	Einkünfte	108,65	108,65
0	-215	Sparbuch	-57	Wohnen	-215	-106,35
16,63	0	Giro	-32,72	Ernährung	16,63	-89,72
25,36	0	Giro	-7,36	Mobilität	25,36	-64,36
0	-215	Giro	-222,36	Wohnen	-430	-279,36
3	0	Sparbuch	-54	Gesundheit	3	-276,36
12	0	Giro	-210,36	Ernährung	28,63	-264,36
0	-7	Giro	-217,36	Ernährung	21,63	-271,36
3,45	0	Giro	-213,91	Wohnen	-426,55	-267,91
13,5	0	Giro	-200,41	Einkünfte	122,15	-254,41
13,5	0	Sparbuch	-40,5	Ernährung	35,13	-240,91

```
SELECT "a".*,
   CASEWHEN( "a"."Betrag" > 0, "a"."Betrag", 0 ) AS "Einnahme",
   CASEWHEN( "a"."Betrag" < 0, "a"."Betrag", 0 ) AS "Ausgabe",
   ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" )
      AS "Kontoname",
   ( SELECT SUM( "Betrag" )
      FROM "Kasse" WHERE "Konto_ID" = "a"."Konto_ID" AND
      ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Saldo_Konto",
   ( SELECT "Kategorie" FROM "Kategorie"
      WHERE "ID" = "a"."Kategorie_ID" )
      AS "Kategoriename",
   ( SELECT SUM( "Betrag" )
      FROM "Kasse" WHERE "Kategorie_ID" = "a"."Kategorie_ID" AND
      ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Saldo_Kategorie",
   ( SELECT SUM( "Betrag" ) FROM "Kasse" WHERE "Datum" < "a"."Datum"
      OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) )
      AS "Saldo_gesamt"
FROM "Kasse" AS "a"
ORDER BY "Datum", "ID" ASC
```

Die äußere Abfrage greift nur auf die Tabelle "Kasse" zu. Der Tabelle wird das Alias "a" zugeschrieben, damit auf die äußere Abfrage mit den korrelierenden Unterabfragen zugegriffen werden kann. Von der Tabelle "Kasse" werden alle Felder übernommen: **`"a".*`** . Die Sortierung wird zuerst nach dem "Datum", dann nach dem Feld "ID" vollzogen.

Sämtliche weiteren korrelierenden Unterabfragen sind bereits vorher in den Teilabfragen aufgetaucht. Die Aufsplittung von "Einnahme" und "Ausgabe" sowie von "Saldo_gesamt" erfolgte in der Abfrage *Saldo*. "Kontoname" und "Saldo_Konto" sind in der Abfrage *Kontoverlauf* teilweise zu finden. "Kategorie" und "Saldo_Kategorie" tauchten entsprechend in der Abfrage *Kategorieverlauf* zum Teil auf. Die letzten beiden Abfragen beziehen sich aber bereits auf die vorher erstellte Ansicht.

Konto_Saldo_Konto_Kategorie_Adressat_Umbuch

In der vorhergehenden Abfrage wurde auf die Berücksichtigung von Umbuchungen verzichtet. Sie eignet sich also nur, wenn tatsächlich in der Datenbank Umbuchungen in zwei nacheinander folgenden Datensätzen erfasst werden.

Einnahme	Ausgabe	Kontoname	Saldo_Konto	Kategoriename	Saldo_Kategorie	Saldo_gesamt
158	0	Sparbuch	158	Einkünfte	158	158
0	-37	Giro	-37	Einkünfte	121	121
0	-12,35	Giro	-49,35	Einkünfte	108,65	108,65
0	-215	Sparbuch	-57	Wohnen	-215	-106,35
16,63	0	Giro	-32,72	Ernährung	16,63	-89,72
25,36	0	Giro	-7,36	Mobilität	25,36	-64,36
0	-215	Giro	-222,36	Wohnen	-430	-279,36
3	0	Sparbuch	-54	Gesundheit	3	-276,36
12	0	Giro	-210,36	Ernährung	28,63	-264,36
0	-7	Giro	-217,36	Ernährung	21,63	-271,36
3,45	0	Giro	-213,91	Wohnen	-426,55	-267,91
13,5	0	Giro	-200,41	Einkünfte	122,15	-254,41
13,5	0	Sparbuch	-40,5	Ernährung	35,13	-240,91

Abfrage «Konto_Saldo_Konto_Kategorie_ Adressat» berücksichtigt keine Umbuchungen.

Der Vergleich mit der nachfolgenden Teilansicht der Abfrage «Konto_Saldo_Konto_Kategorie_Adressat_Umbuch» zeigt zusätzliche Spalten für den "Umbuch_Kontoname" und den "Saldo_Umbuch_Konto". Da lediglich zwei Beträge umgebucht wurden steht entsprechend auch nur in zwei Feldern ein Betrag unterschiedlich von 0, beides Mal für das Konto 'Bargeld'.

Bereits beim Betrachten der zweiten Datenzeile fällt aber daneben auf, dass sich weder "Saldo_Kategorie" noch "Saldo_gesamt" gegenüber dem ersten Datensatz verändert haben. Die Umbuchungen verschiebt nur das Geld. Ausgegeben und damit auch als Ausgabe verbucht wird der Betrag nicht. Entsprechend stimmen die letzten beiden Spalten auch in der Summe im letzten Datensatz nicht überein. Der Saldo des Umbuch_Kontos beträgt jetzt 252,- €, die als Bargeld im Portemonnaie schlummern. Entsprechend fällt der Kassensturz in "Saldo_gesamt" auch um 252,- € günstiger aus.

Saldo_Konto	Umbuch_Kontoname	Saldo_Umbuch_Konto	Kategoriename	Saldo_Kategorie	Saldo_gesamt
158		0	Einkünfte	158	158
-37	Bargeld	37	Einkünfte	158	158
-49,35		0	Einkünfte	145,65	145,65
-57		0	Wohnen	-215	-69,35
-32,72		0	Ernährung	16,63	-52,72
-7,36		0	Mobilität	25,36	-27,36
-222,36	Bargeld	252	Wohnen	-215	-27,36
-54		0	Gesundheit	3	-24,36
-210,36		0	Ernährung	28,63	-12,36
-217,36		0	Ernährung	21,63	-19,36
-213,91		0	Wohnen	-211,55	-15,91
-200,41		0	Einkünfte	159,15	-2,41
-40,5		0	Ernährung	35,13	11,09

Abfrage «Konto_Saldo_Konto_Adressat_Umbuch» mit der Berücksichtigung von Umbuchungen in "Saldo_Kategorie" und "Saldo_gesamt".

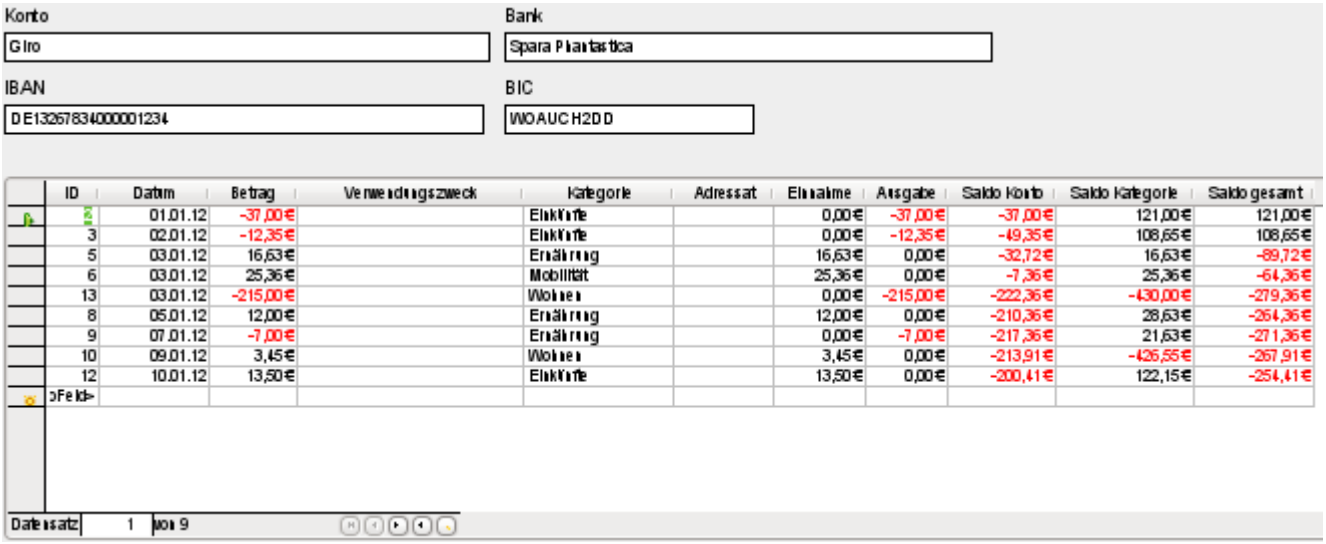
```
SELECT "a".*,
   CASEWHEN( "a"."Betrag" > 0, "a"."Betrag", 0 ) AS "Einnahme",
   CASEWHEN( "a"."Betrag" < 0, "a"."Betrag", 0 ) AS "Ausgabe",
   ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Konto_ID" )
      AS "Kontoname",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Buchungen"
      WHERE "Konto_ID" = "a"."Konto_ID" AND ( "Datum" < "a"."Datum"
      OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Saldo_Konto",
   ( SELECT "Konto" FROM "Konto" WHERE "ID" = "a"."Umbuch_Konto_ID" )
      AS "Umbuch_Kontoname",
   IFNULL( ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Buchungen"
      WHERE "Konto_ID" = "a"."Umbuch_Konto_ID"
      AND ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) ), 0 )
      AS "Saldo_Umbuch_Konto",
   ( SELECT "Kategorie"
      FROM "Kategorie" WHERE "ID" = "a"."Kategorie_ID" )
      AS "Kategoriename",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Buchungen"
      WHERE "Kategorie_ID" = "a"."Kategorie_ID"
      AND ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Saldo_Kategorie",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Buchungen"
      WHERE ( "Datum" < "a"."Datum" OR
      ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Saldo_gesamt"
FROM "Kasse" AS "a"
ORDER BY "Datum", "ID" ASC
```

Das Prinzip gleicht der vorhergehenden Abfrage. Bei den Summenermittlungen werden aber immer die Summen berücksichtigt, die durch eine Berücksichtigung des Umbuchwertes zustande kommen, da sämtliche Werte aus der Ansicht ermittelt werden. Siehe hierzu auch die Abfragen *Saldo_Konto* und *Saldo_Kategorie* sowie die *Ansicht_Kasse_mit_Umbuchungen.*

Formulare

Letztlich wird für eine funktionierende Datenbank in diesem Fall nur ein Formular benötigt. Die im Folgenden vorgestellten Formulare zeigen die verschiedenen Zugriffsweisen auf die Daten auf und sind entsprechend für unterschiedliche Belange optimiert.

Konto



ID	Datum	Betrag	Verwendungszweck	Kategorie	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo Kategorie	Saldo gesamt
2	01.01.12	-37,00€		Einkäufe		0,00€	-37,00€	-37,00€	121,00€	121,00€
3	02.01.12	-12,35€		Einkäufe		0,00€	-12,35€	-49,35€	108,65€	108,65€
5	03.01.12	16,63€		Ernährung		16,63€	0,00€	-32,72€	16,63€	-89,72€
6	03.01.12	25,36€		Mobilität		25,36€	0,00€	-7,36€	25,36€	-64,36€
13	03.01.12	-215,00€		Wohnen		0,00€	-215,00€	-222,36€	-430,00€	-279,36€
8	05.01.12	12,00€		Ernährung		12,00€	0,00€	-210,36€	28,63€	-264,36€
9	07.01.12	-7,00€		Ernährung		0,00€	-7,00€	-217,36€	21,63€	-271,36€
10	09.01.12	3,45€		Wohnen		3,45€	0,00€	-213,91€	-426,55€	-267,91€
12	10.01.12	13,50€		Einkäufe		13,50€	0,00€	-200,41€	122,15€	-254,41€
>Feld<										

Datensatz 1 von 9

Im Hauptformular ist die Datengrundlage die Tabelle «Konto». Hier können durch diese Formularkonstruktion also neue Konten eingefügt werden.

Im Unterformular ist die Abfrage «Konto_Saldo_Konto_Kategorie_Adressat» als Datengrundlage festgelegt. Das Unterformular ist mit dem Hauptformular über den Fremdschlüssel des Feldes «Konto_ID» mit dem Primärschlüssel «ID» aus dem Hauptformular verbunden.

Die Felder «Kategorie» und «Adressat» sind Listenfelder. Hier wird die Bezeichnung aus den dazugehörigen Tabellen ausgelesen und der Fremdschlüssel in die Abfrage des Unterformulars eingetragen. Alle dem Feld «Adressat» folgenden Felder sind berechnete Felder, können also keine Eingabe aufnehmen. Sie sind deshalb folgendermaßen eingestellt: **Eigenschaften → Allgemein → Aktiviert → Nein** und **Eigenschaften → Allgemein → Nur Lesen → Ja**. Werden Datenfelder, wie häufig bei Formularen üblich, mit der Tastatur über Tabulatoren angesprungen, so zeigt sich bei diesem Tabellenkontrollfeld leider ein recht ärgerlicher Bug: Nach Eingabe eines Adressaten und Drücken des Tabulators verschwindet der Cursor. Die Tabulatortaste muss jetzt 5 Mal gedrückt werden bis der Cursor wieder am Start der nächsten Zeile auftaucht. Dies liegt daran, dass sich innerhalb eines Tabellenkontrollfeldes leider der Tabulator nicht abstellen lässt. Dies ist gemeldet als *Bug 43928* .

Kasse

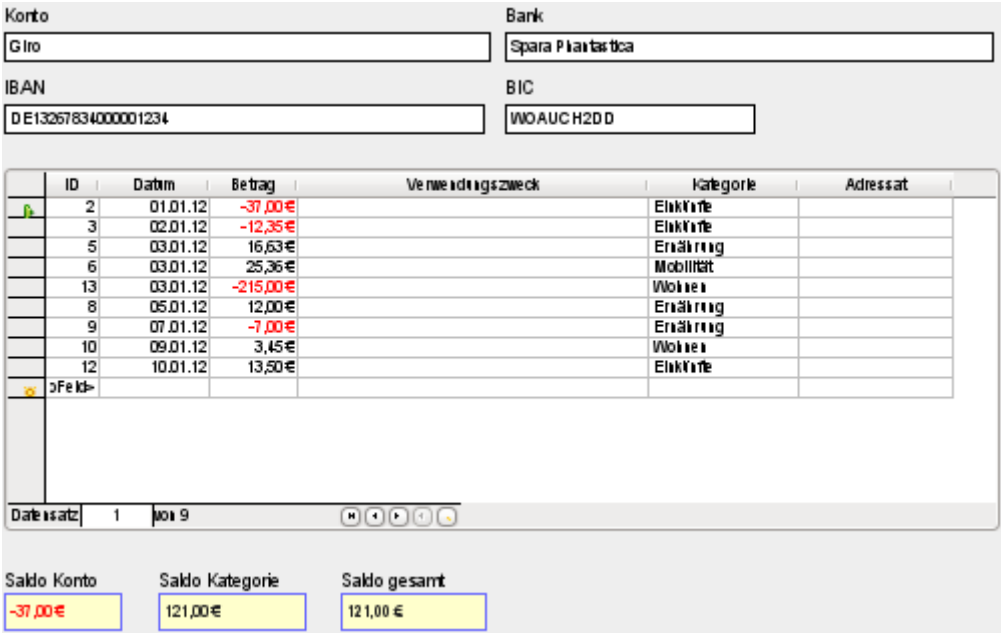
Konto	Datum	Betrag	Verwendungszweck	Kategorie	Adressat	Einnahme	Ausgabe	Saldo Konto	Saldo Kategorie	Saldo gesa
Spark	01.01.12	158,00€		Einkäufe		158,00€	0,00€	158,00€	158,00€	158,0
Giro	01.01.12	-37,00€		Einkäufe		0,00€	-37,00€	-37,00€	121,00€	121,0
Giro	02.01.12	-12,35€		Einkäufe		0,00€	-12,35€	-49,35€	108,65€	108,6
Sparbuch	03.01.12	-215,00€		Wohnen		0,00€	-215,00€	-57,00€	-215,00€	-106,3
Giro	03.01.12	16,63€		Ernährung		16,63€	0,00€	-32,72€	16,63€	-89,7
Giro	03.01.12	25,36€		Mobilität		25,36€	0,00€	-7,36€	25,36€	-64,3
Giro	03.01.12	-215,00€		Wohnen		0,00€	-215,00€	-222,36€	-430,00€	-279,3
Sparbuch	04.01.12	3,00€		Gesundheit		3,00€	0,00€	-54,00€	3,00€	-276,3
Giro	05.01.12	12,00€		Ernährung		12,00€	0,00€	-210,36€	28,63€	-264,3
Giro	07.01.12	-7,00€		Ernährung		0,00€	-7,00€	-217,36€	21,63€	-271,3
Giro	09.01.12	3,45€		Wohnen		3,45€	0,00€	-213,91€	-426,55€	-267,9
Giro	10.01.12	13,50€		Einkäufe		13,50€	0,00€	-200,41€	122,15€	-254,4
Sparbuch	12.01.12	13,50€		Ernährung		13,50€	0,00€	-40,50€	35,13€	-240,9

Datensatz 1 von 13

Das Formular «Kasse» besteht nur aus einem einzigen Tabellenkontrollfeld in einem Formular, das als Datengrundlage auf die Abfrage «Konto_Saldo_Konto_Kategorie_Adressat» zugreift. Das Konto wird direkt in dem Tabellenkontrollfeld über ein Listenfeld ausgewählt. Dafür ist einfach das Feld «ID» entsprechend geändert worden. Das Feld «ID» ist für die Eingabe nicht nötig, da der Wert dafür schließlich automatisch erstellt wird.

Vorteil dieses Formulars ist, dass jetzt direkt alle Konten verwaltet werden können. Alle Datensätze aus der Tabelle «Kasse» sind zu sehen. In so einem Formular könnten also auch Umbuchen ohne besondere Probleme durchgeführt werden, indem der darüber liegende Datensatz einfach kopiert wird und dann ein anderes Konto sowie das entsprechende Vorzeichen für den Betrag gesetzt wird. Dann ist es nicht notwendig, separat in der Tabelle «Kasse» eine «Umbuch_Konto_ID» zu führen.

Konto_Salden_separat

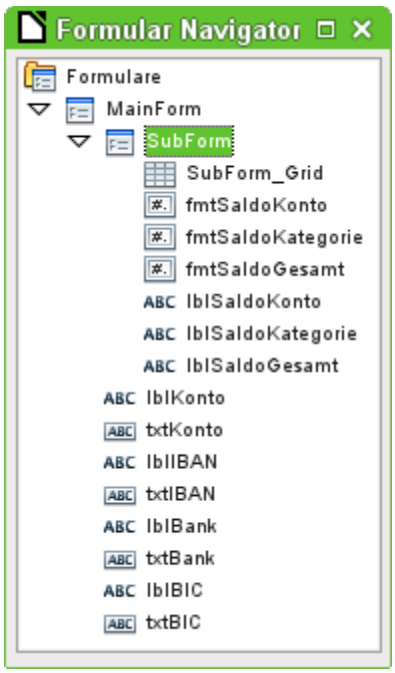


Das Formular «Konto_Salden_separat» zeigt den ersten Zugriff, wie mit einfachsten Mitteln die Salden, die vorher in dem Tabellenkontrollfeld steckten, ausgelagert werden können. In dem Tabellenkontrollfeld führten sie zum einen zu einem sehr breites Tabellenkontrollfeld, zum anderen trugen sie durch den Tabulatorbug auch noch zum vorübergehenden Verschwinden des Cursors bei.

Dieses Formular baut auf die gleichen Datengrundlagen auf, auf die auch das Formular «Konto» aufbaut. Es zeigt unter dem Tabellenkontrollfeld die Salden an, die sonst bei dem entsprechenden Datensatz in dem Tabellenkontrollfeld weiter rechts stehen. Da der Cursor nicht aus dem Tabellenkontrollfeld heraus springt wird so vermieden, dass unnötige Tabulatorbetätigungen erfolgen müssen.

Leider wird auf diese Art und Weise nur der Saldo für das jeweils aktuelle Konto und die jeweils aktuelle Kategorie zu dem jeweils aktuellen Zeitpunkt ermittelt. Informationen zu anderen Konten und Kategorien sind nicht möglich.

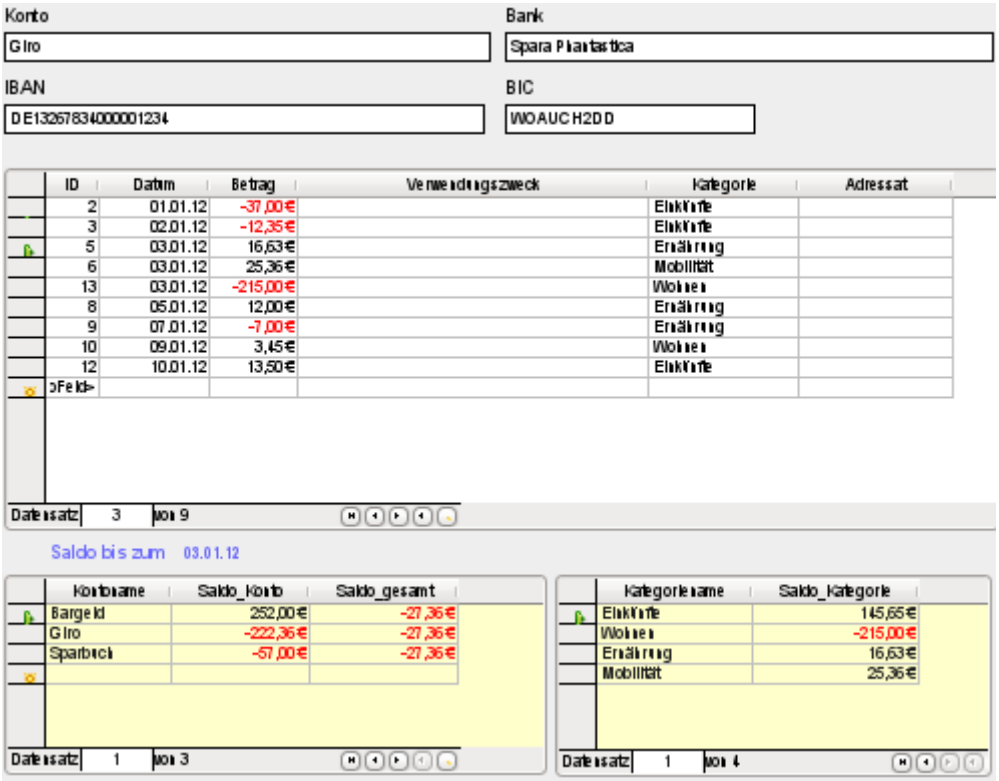
Wird ein neuer Datensatz eingegeben, so bleiben die Informationsfelder außerdem leer. Zu dem dort aktuellen Konto und der dort aktuellen Kategorie fehlt also immer jeder Hinweis.



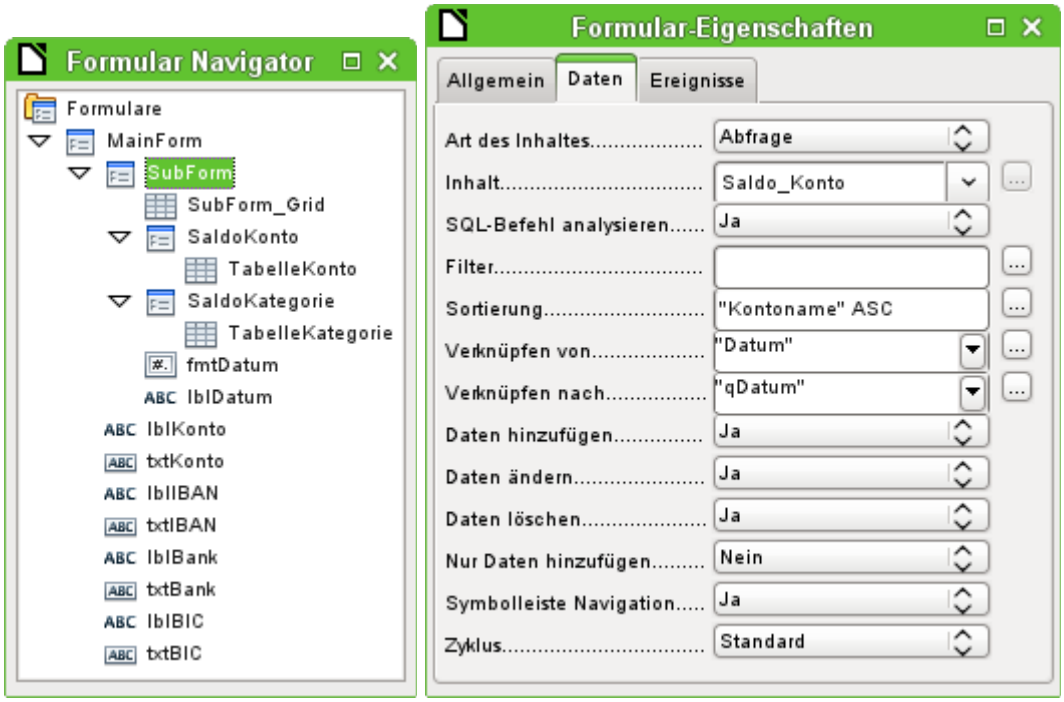
Wie so eine Auslagerung von Feldern aus dem Tabellenkontrollfeld erstellt wird zeigt der Formularnavigator. Hier finden sich in dem Unterformular neben dem Tabellenkontrollfeld «SubForm_Grid» die formatierbaren Feld für die verschiedenen Salden. Die formatierbaren Felder sind hier die bevorzugten Felder zur Währungsdarstellung, da sie auch die Möglichkeit bieten, negative Beträge in Rot erscheinen zu lassen. Das ist bei reinen Währungsfeldern leider nicht möglich.

Konto_Salden

In diesem Formular werden die Salden aller Konten und Kategorien aufgezeigt. Das dargestellt Saldo gilt aber auch nur bis zu dem Datum, das in dem aktuellen Datensatz aus dem Tabellenkontrollfeld enthalten ist.



Mit dieser Saldo-Angabe ist es also schon besser möglich, einen entsprechenden Überblick zu erhalten.



Unterhalb des ursprünglichen Unterformulars «SubForm», das auf **Daten → Art des Inhaltes → Tabelle** und **Inhalt → Kasse** umgestellt wird, sind zwei weitere Formulare angegliedert. Das eine Formular stellt den Saldo für alle Kontoarten dar, das andere Formular den Saldo für alle Kategorien.

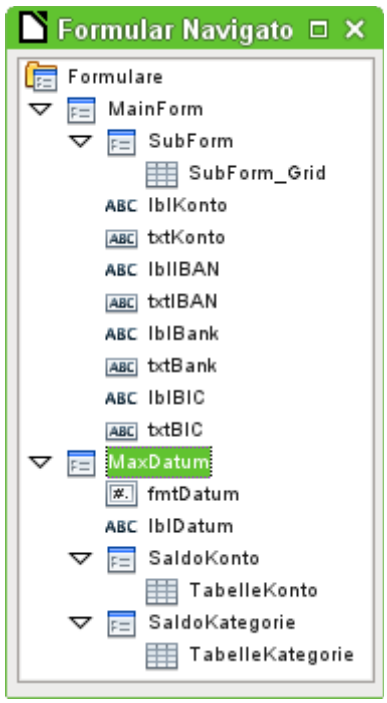
Beide Unterformulare beruhen jeweils auf Abfragen; das Unterformular «SaldoKonto» auf der Abfrage *Saldo_Konto* und das Unterformular «SaldoKategorie» auf der Abfrage *Saldo_Kategorie* . Beide Abfragen sind Parameterabfragen. Der Parameter für das Datum lautet bei beiden Abfragen «qDatum». Entsprechend sind auch die Unterformulare mit dem darüber liegenden Formular verknüpft: "Datum" aus dem Formular «SubForm» ist im Unterformular «SaldoKonto» "qDatum", ebenso im Unterformular «SaldoKategorie».

Nachteil dieser Konstruktion ist, dass zwar der aktuelle Kontostand zu dem jeweiligen Datum angezeigt wird, auch immer schön aktuell zu dem jeweiligen Datensatz. Bei einer Neueingabe ist davon aber leider nichts zu sehen. Schließlich ist bei neuen Datensätzen noch kein "Datum" im Formular «SubForm» vorhanden, das an die Unterformulare weiter gegeben werden kann. Diese Konstruktion ist also zur Recherche sinnvoll. Für die Neueingabe von Daten wäre eine Anzeige unabhängig von dem Formular «SubForm» sinnvoller.

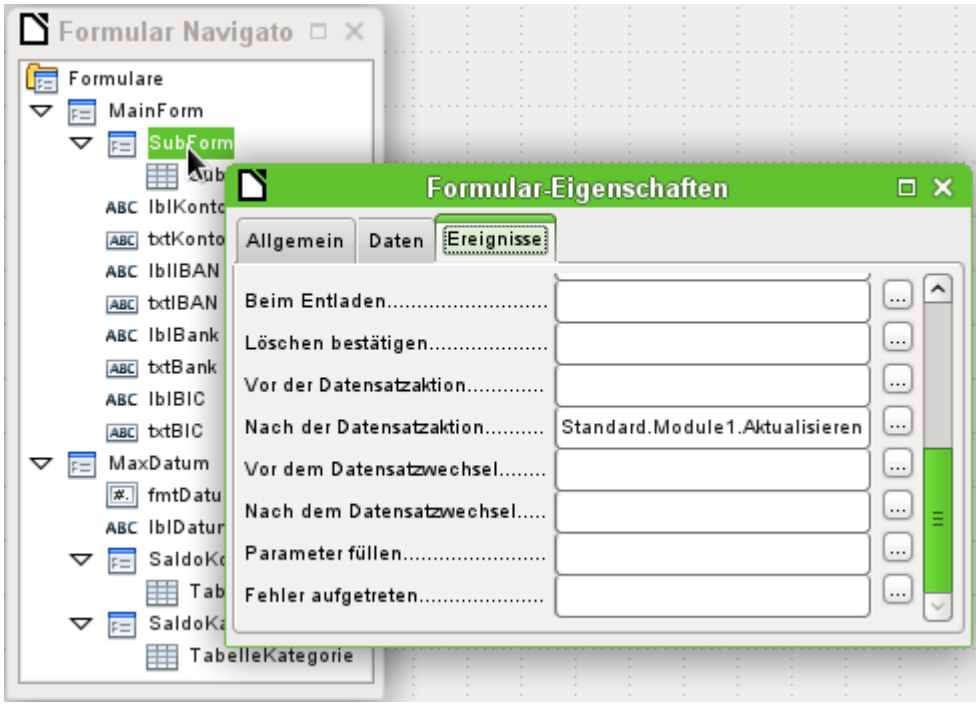
Konto_Salden_komplett

Das Formular «Konto_Salden_komplett» bewirkt, dass jetzt unabhängig von dem Unterformular mit dem Tabellenkontrollfeld die entsprechenden Salden angezeigt werden. Außerdem ist die Anzeige auf alle Datensätze eingestellt, die bisher gespeichert wurden, zeigt also immer den zur Eingabezeit aktuellen Kontostand an.

Das grundsätzliche Erscheinungsbild ist gleich dem vorherigen Formular, lediglich das aufgezeigte Datum ist eben immer das Datum, das den höchsten Datumswert aus der tabelle «Kasse» wiedergibt.



Das Formular «MaxDatum» liegt neben dem Formular «MainForm». Die beiden Formulare sind zwar auf einer gemeinsamen Oberfläche zu sehen, haben aber sonst miteinander keine Verbindung. Datenquelle für das Formular «MaxDatum» ist die Abfrage *Datum_Max* . Die beiden darunter liegenden Formulare beruhen wieder auf den gleichen Abfragen wie im vorhergehenden Formular, haben entsprechend auch die gleichlautende Verknüpfung.



Nachteil dieser Konstruktion ist, dass das Nebenformular «MaxDatum» nicht mit bekommt, wenn im Unterformular «SubForm» ein Datensatz geändert wird, der entsprechend Auswirkungen auf die dargestellten Werte hat. Das könnte mit einem Button zum Aktualisieren der Daten behoben werden. Eleganter ist es aber, dies durch ein einfaches Makro erledigen zu lassen, das in dem Formular «SubForm» mit dem Ereignis **Nach der Datensatzaktion** verbunden wird:

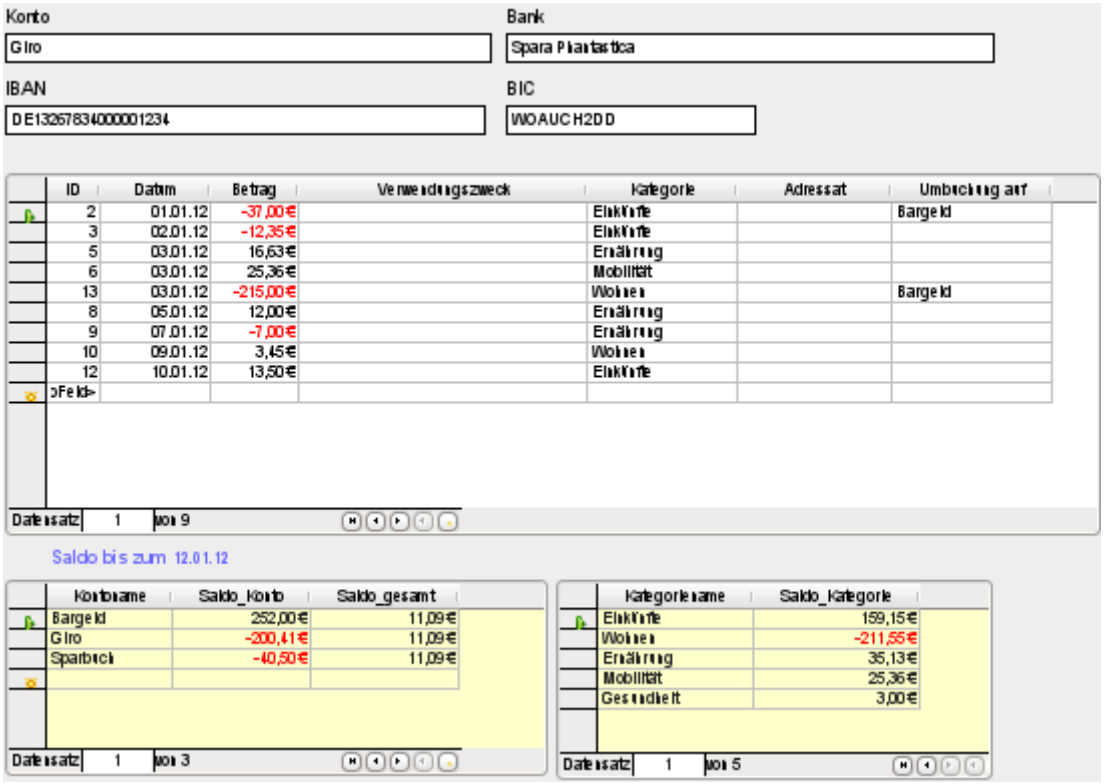
```
SUB Aktualisieren
   DIM oDoc AS OBJECT
```

```
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MaxDatum")
    oForm.reload
END SUB
```

Das Formular «MaxDatum» liegt auf der gleichen Formularoberfläche wie das Formular «SubForm». Entsprechend wird das Formular direkt über **`thisComponent.drawpage`** angesteuert. Das Formular wird mit dem Namen benannt, so dass das korrekte Formular auch gefunden wird. Anschließend wird das Formular einfach neu geladen.

Damit wird bei jeder den Datensatz in dem Formular «SubForm» ändernden Aktion entsprechend ein Neuladen der Übersicht über die kompletten Salden veranlasst.

Konto_Salden_komplett_Umbuchung



Hier wird schließlich als einzige Ergänzung noch die Möglichkeit geboten, auch die Umbuchungen in das Tabellenkontrollfeld mit auf zu nehmen. Ansonsten ist das Formular identisch mit dem vorhergehenden, zeigt also immer die aktuellen Salden, nicht aber den Kontostand zu dem jeweils vermerkten Datensatz, an.

Berichte

In den beiden Berichten soll dargestellt werden, welche Buchungen im Laufe der Zeit erfolgt sind. Der Bericht fragt dabei alle Buchungen ab. Dies ist natürlich dann nicht mehr sinnvoll, wenn die Datenbank deutlich mehr mit Daten gefüllt ist. Wie eine entsprechende Vorfilterung des Berichtes nach den Datumsangaben erfolgen kann zeigt daher die folgende erweiterte Datenbank auf.

Die prinzipielle Konstruktion der Berichte in beiden Datenbank und auch für Kategorien und Konten ist allerdings gleich:

Die Datenbasis für den Bericht ist eine Abfrage, in dem obigen Beispiel die Abfrage «Kategorieverlauf». Seitenkopf und Seitenfuß wurden entfernt. Auch einen Berichtkopf oder einen Berichtsfuß gibt es nicht. Damit der Bericht die Kategorien zusammen gruppiert ist eine Gruppe «Kategorie» erstellt worden. Um innerhalb dieser Gruppe eine Sortierung nach dem Datum zu erreichen wird zusätzlich eine Gruppe für das Datum erstellt. Diese Gruppe erscheint nicht im Bericht, da in den **Eigenschaften→Allgemein → Sichtbar → Nein** gewählt wurde.

In dem Abschnitt «Detail» werden nacheinander alle Daten für die jeweilige Kategorie eingelesen. Die Beschriftungen wurden dabei aber in den Gruppenkopf der Kategorie verlegt, damit sie nicht auch wiederholt werden.

Bei den Feldern für den Betrag und für den Kategorieverlauf muss noch unter **Eigenschaften→Allgemein →Formatierung** das Feld auf die Anzeige der Währung umgestellt werden. Hier scheint es manchmal Probleme zu geben, so dass der Report-Builder anschließend nur leere Felder anzeigt. Beim Erstellen der Beispieldatenbank wurde zwischendurch die Version von LO gewechselt. Anschließend klappte auch die Anzeige der Währung.

Für die Kategorie wird auch ein Gruppenfuß erstellt. In diesem Gruppenfuß wird der Verlauf noch einmal in einem Diagramm dargestellt. Dabei sind die Einträge nach Jahr und Monat gruppiert. Es wird also sozusagen ein monatlicher Abschluss zu den Kategorien dargestellt.

Um so ein Diagramm zu erzeugen wird zuerst über die Symbolleiste Diagramm gewählt oder aus dem Menü heraus **Einfügen → Bericht-Steuerelemente → Diagramm** gestartet. Das Diagramm wird in der gewünschten Größe aufgezogen. Anschließend wird unter **Eigenschaften → Daten → Art des Inhaltes** auf den gewünschten Inhalt eingestellt. In dem obigen Fall war dies die Abfrage, genauer die Abfrage «monatlich_Kategorie». Aus dem Bericht wird das Feld "Kategorie" mit dem Feld "Kategorie" des Diagramms verknüpft. Es werden also nur die Werte im Diagramm angezeigt, die mit der Kategorie der jeweiligen Gruppe zusammen hängen.

Unter **Eigenschaften → Allgemein → Diagrammtyp** kann jetzt noch eingestellt werden, mit welchem Diagrammtyp denn nun die Daten dargestellt werden. Hier wurde der Typ **Säulendiagramm → Gestapelt** gewählt. Beim normalen Säulendiagramm wurde die Säule nicht mittig dargestellt. Der Text war dann etwas nach links verschoben – so als ob nicht ein sondern zwei Einträge in der Säule erfolgt wären.

Weitere Einstellungsmöglichkeiten zu Achsen usw. sind über einen Doppelklick auf das Diagramm zugänglich.

Fortlaufendes Saldo mit Splitbuchungen

Großeinkauf und keine Übersicht mehr? Waren das jetzt alles Lebensmittel, oder ist da nicht auch noch etwas für die Sparte «Hobby» dabei gewesen? In dieser Version «Beispiel_Saldo_fortlaufend_Splitbuchung» können Einnahmen und Ausgaben zuerst einmal auf einem Konto verbucht und dann auf verschiedene Kategorien verteilt werden.

Diese Datenbank erfordert deutlich mehr Aufwand auch in der Makroprogrammierung, wenn sie denn möglichst fehlerfrei zu bedienen sein soll. Sonst passen eventuell die Ausgaben des Großeinkaufs nicht mehr zu der entsprechenden Aufteilung der Kategorien.

Tabellen

Die Tabelle "Kasse" ist gegenüber der Tabelle aus «Beispiel_Saldo_fortlaufend» leicht verändert worden. In der Hauptsache wurde das Feld "Kategorien_ID" entfernt. Hinzugekommen ist ein Feld, das periodisch wiederkehrende Buchungen kennzeichnen soll.

In der Tabelle "Kasse" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Betrag	Zahl	Der Betrag, der aus dem folgenden Konto abgebucht oder hinzu gebucht wird. Bei der Felderstellung ist auf 2 Nachkommastellen zu achten. **Feldeigenschaften → Eingabe erforderlich → Ja**
Konto_ID	Tiny Integer	Fremdschlüssel aus der Tabelle "Konto". **Feldeigenschaften → Eingabe erforderlich → Ja**
Datum	Datum	Hier soll das Datum gespeichert werden, zu dem die Einnahme/Ausgabe erfolgt ist. **Feldeigenschaften → Eingabe erforderlich → Ja**
Adressat_ID	Integer	Fremdschlüssel aus der Tabelle "Adressat".
Verwendungszweck	Text	Wofür ist das Geld verwandt worden? Dieses Feld speichert Zusatzinformationen zu den Kategorien.
Umbuch_Konto_ID	Tiny Integer	Wird Geld z.B. vom Girokonto abgehoben, so wechselt das Geld lediglich das Konto von «Giro» zu «Bargeld», ist aber noch nicht anderweitig ausgegeben.
periodisch	Ja/Nein	Wenn eine Buchung häufiger in ähnlicher Art und Weise vorkommt, so soll sie in diesem Feld gekennzeichnet werden. Diese Buchungen können in einem Formular als neue Datensätze bis auf das Datum übernommen werden.

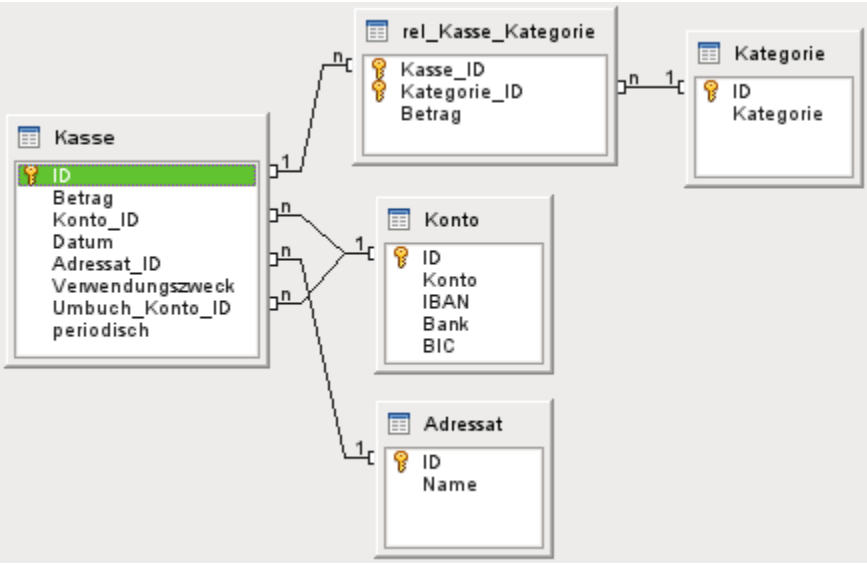
Der **`Constraint`** dieser Tabelle ist genauso definiert wie bei der vorhergehenden Datenbank «Beispiel_Saldo_fortlaufend».

Die Tabellen "Kategorie", "Konto" und "Adressat" sind ebenfalls identisch. Die Tabelle "Kategorie" ist allerdings jetzt in einer n:m-Beziehung zur Tabelle "Kasse" definiert, so dass einer Buchung mehrere Kategorien zugeordnet werden können, aber auch einer Kategorie mehrere Buchung zugeordnet werden können.

In der Tabelle "rel_Kasse_Kategorie" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
Kasse_ID	Integer	Zusammen mit "Kategorie_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur Tabelle "Kasse".
Kategorie_ID	Tiny Integer	Zusammen mit "Kasse_ID" Primärschlüssel der Tabelle. Gleichzeitig Fremdschlüssel für die Verbindung zur Tabelle "Kategorie".
Betrag	Zahl	Der Betrag, der aus dem folgenden Konto abgebucht oder hinzu gebucht wird. Bei der Felderstellung ist auf 2 Nachkommastellen zu achten. **Feldeigenschaften → Eingabe erforderlich → Ja**

Die Tabellen werden unter **Extras → Beziehungen** wie folgt verknüpft:



Schließlich gibt es noch die Tabelle "Filter", die allerdings nicht zu den anderen Tabelle in Beziehung steht. Sie dient hier lediglich dazu, periodische Buchungen zu übertragen.

In der Tabelle "Filter" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
ID	Ja/Nein	Primärschlüssel der Tabelle. Diese Tabelle nimmt nur einen Datensatz auf.
Kasse_ID	Integer	Zwischenspeicherfeld für den Wert des Primärschlüssel aus "Kasse". Dieses Feld wird für die periodische Buchung benötigt.
StartDatum	Datum	Zwischenspeicher für das Datum, von dem ab der Bericht erstellt werden soll.
EndDatum	Datum	Zwischenspeicher für das Datum, bis zu dem der Bericht erstellt werden soll
DatensatzDatum	Datum	Zwischenspeicher für das jeweilige Datum des Datensatzes, der im Formular gerade bearbeitet wird. Wird bei einem neuen Datensatz mit dem aktuellen Datum zur Zeit der Neueingabe versehen.

Die Tabelle "Filter" wird zum Start mit einem Datensatz versehen. Das Feld "ID" wird dazu auf 'Ja' eingestellt.

Diese Tabelle kann noch erweitert werden, so dass sie auch zum Filtern von Daten dienen kann.

Abfragen

Einige der Abfrage sind identisch zu den Abfragen, die bereits in der Datenbank «Beispiel_Saldo_fortlaufend.odb» enthalten sind.

Saldo

Siehe *Saldo*.

Ansicht_Kasse_mit_Umbuchungen

Die Ansicht aus der vorhergehenden Datenbank *Ansicht_Kasse_mit_Umbuchungen* enthält auch eine Spalte für die Kategorie bzw. die Kategorie_ID. Diese beiden Spalten müssen in der Ansicht dieser Datenbank fehlen, da die Tabelle «Kategorie» nicht mehr über eine 1:n-Beziehung mit der Tabelle «Kasse» verbunden ist. Der weitere Inhalt der Abfrage ist allerdings identisch.

Ansicht_Kasse_Kategorie

Diese Ansicht ist notwendig, da jetzt ohne weiteres mehrere Kategoriezuweisungen zu einem Buchungsvorgang möglich sind.

ID	Gesamtbetrag	Splitbetrag	Konto	Datum	Name	Verwendungszweck	Kategorie
1	1.685,34 €	1.685,34 €	Giro	01.01.14		Gehalt	Einkünfte
3	-12,69 €	-12,69 €	Bargeld	02.01.14		Medikamente	Gesundheit
4	-215,00 €	-91,00 €	Sparbuch	10.01.14			Mobilität
4	-215,00 €	-124,00 €	Sparbuch	10.01.14			Hobby
5	-16,64 €	-16,64 €	Giro	15.01.14			Mobilität
6	1.685,34 €	1.685,34 €	Giro	03.02.14		Gehalt	Einkünfte
7	13,00 €	13,00 €	Sparbuch	04.02.14			Einkünfte
8	-129,95 €	-129,95 €	Giro	05.02.14		Joggingschuhe	Hobby
9	-8,50 €	-8,50 €	Giro	17.02.14		Laufsocken	Hobby
10	-45,37 €	-45,37 €	Bargeld	09.01.14		Markt, Lebensmittel	Ernährung
11	-45,57 €	-32,07 €	Bargeld	12.01.14		Markt, Lebensmittel	Ernährung
11	-45,57 €	-13,50 €	Bargeld	12.01.14		Markt, Lebensmittel	Hobby
12	-250,00 €	-88,00 €	Giro	10.01.14		Versicherungen etc.	Wohnen
12	-250,00 €	-75,60 €	Giro	10.01.14		Versicherungen etc.	Ernährung
12	-250,00 €	-86,40 €	Giro	10.01.14		Versicherungen etc.	Gesundheit

Die "ID" 4 ist doppelt vertreten, die "ID" 11 ebenfalls, die "ID 12 sogar dreifach. Hier sind die jeweiligen Gesamtbeträge in Splitbeträge aufgegliedert worden.

```
SELECT "Kasse"."ID", "Kasse"."Betrag" AS "Gesamtbetrag",
   IFNULL( "rel_Kasse_Kategorie"."Betrag", "Kasse"."Betrag" )
   AS "Splitbetrag", "Konto"."Konto", "Kasse"."Datum",
   "Adressat"."Name", "Kasse"."Verwendungszweck",
   "rel_Kasse_Kategorie"."Kategorie_ID",
   IFNULL( "Kategorie"."Kategorie", '- keine Angabe -' )
   AS "Kategorie"
FROM "Kasse"
   LEFT JOIN "Konto" ON "Kasse"."Konto_ID" = "Konto"."ID"
   LEFT JOIN "Adressat" ON "Kasse"."Adressat_ID" = "Adressat"."ID"
   LEFT JOIN "rel_Kasse_Kategorie"
      ON "Kasse"."ID" = "rel_Kasse_Kategorie"."Kasse_ID"
   LEFT JOIN "Kategorie"
      ON "rel_Kasse_Kategorie"."Kategorie_ID" = "Kategorie"."ID"
WHERE "Kasse"."Umbuch_Konto_ID" IS NULL
```

Zu jedem Eintrag aus der Tabelle "Kasse" muss auf jeden Fall ein Eintrag in dieser Ansicht auftauchen, damit die Beträge aus den Kategorien auch den Beträgen der Kasse entsprechen. Deshalb sind alle Tabellen miteinander über **`LEFT JOIN`** mit der Tabelle "Kasse" verbunden. Über **`IFNULL( "Kategorie"."Kategorie", '- keine Angabe -' )`** wird aufgezeigt, wo vielleicht einem Betrag aus der Kasse keine Kategorie zugewiesen wurde. Dies ist notwendig, wenn die entsprechende Absicherung nicht innerhalb des Formulars erfolgt.

Alle Datensätze, bei denen es sich um eine Umbuchung handelt, werden nicht berücksichtigt. Selbst wenn dort eine Kategoriezuweisung stattfinden würde, dann würde der gleiche Betrag zuerst Addiert und dann subtrahiert.

Kontoverlauf

Der Kontoverlauf ist im Gegensatz zu der vorhergehenden Datenbank jetzt noch nach dem Datum gefiltert, damit entsprechend auch der Bericht über einen begrenzten Zeitraum erstellt werden kann.

```
SELECT "a"."Konto", "a"."Datum", "a"."Betrag",
   ( SELECT SUM( "Betrag" ) FROM "Ansicht_Kasse_mit_Umbuchungen"
      WHERE "Konto" = "a"."Konto" AND ( "Datum" < "a"."Datum"
      OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Kontoverlauf",
   ( SELECT "StartDatum" FROM "Filter" WHERE "ID" = TRUE )
      AS "StartDatum",
   ( SELECT "EndDatum" FROM "Filter" WHERE "ID" = TRUE ) AS "EndDatum"
FROM "Ansicht_Kasse_mit_Umbuchungen" AS "a"
WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
ORDER BY "a"."Konto" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Die Anzeige des in der Tabelle "Filter" verzeichneten "StartDatum" und "EndDatum" wird für den Bericht benötigt. Im Bericht soll auch erscheinen, für welchen Zeitraum er gilt.

Die Filterung der Datensätze erfolgt nach den beiden Datumswerten mit Hilfe des Befehls **`WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"`**. Die Datumswerte **`"StartDatum"`** und **`"EndDatum"`** sind der Abfrage durch den Alias **`AS "StartDatum"`** bzw. **`AS "EndDatum"`** innerhalb der Abfrage bereits bekannt.

Kategorieverlauf

Gegenüber der vorhergehenden Datenbank muss hier doch stärker nachgebessert werden, da die Tabelle «Kategorien» gegenüber der Tabelle «Kasse» in einem n:m-Verhältnis steht. Deshalb wurde zuerst die Ansicht «Ansicht_Kasse_Kategorie» erstellt.

```
SELECT "a"."Kategorie", "a"."Datum", "a"."Splitbetrag" "Betrag",
   ( SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie"
      WHERE "Kategorie" = "a"."Kategorie" AND ( "Datum" < "a"."Datum"
      OR ( "Datum" = "a"."Datum" AND "ID" <= "a"."ID" ) ) )
      AS "Kategorieverlauf",
   ( SELECT "StartDatum" FROM "Filter" WHERE "ID" = TRUE )
      AS "StartDatum",
   ( SELECT "EndDatum" FROM "Filter" WHERE "ID" = TRUE ) AS "EndDatum"
FROM "Ansicht_Kasse_Kategorie" AS "a"
WHERE "a"."Datum" BETWEEN "StartDatum" AND "EndDatum"
ORDER BY "a"."Kategorie" ASC, "a"."Datum" ASC, "a"."ID" ASC
```

Ansonsten unterscheidet sich die Abfrage vom Prinzip her nicht von der Abfrage «Kontoverlauf». Auch diese Abfrage wird nur für den Bericht benötigt.

Saldo_Konto

Siehe *Saldo_Konto* .

Saldo_Kategorie

Die Abfrage zur Ermittlung des Saldos der Kategorie erfolgt über die bereits erstellte "Ansicht_Kasse_Kategorie".

```
SELECT "Kategorie_ID", "Kategorie",
   SUM( "Splitbetrag" ) AS "Saldo_Kategorie",
   ( SELECT SUM( "Splitbetrag" ) FROM "Ansicht_Kasse_Kategorie"
   WHERE "Datum" <= :qDatum ) AS "Saldo_gesamt"
FROM "Ansicht_Kasse_Kategorie"
WHERE "Datum" <= :qDatum
GROUP BY "Kategorie", "Kategorie_ID"
ORDER BY "Kategorie" ASC
```

Eine einfache Abfrage, zusammengestellt über die grafische Benutzeroberfläche. Lediglich die Unterabfrage zur Ermittlung von "Saldo_gesamt" musste in der SQL-Ansicht hinzu gefügt werden.

Datum_Max

Siehe *Datum_Max*.

monatlich_Konto

```
SELECT YEAR( "Datum" ) || '-' || RIGHT( '0' || MONTH( "Datum" ), 2 )
   AS "JahrMonat", "Konto", SUM( "Betrag" ) AS "Summe"
FROM "Ansicht_Kasse_mit_Umbuchungen"
WHERE "Datum"
BETWEEN ( SELECT "StartDatum" FROM "Filter" WHERE "ID" = TRUE )
AND ( SELECT "EndDatum" FROM "Filter" WHERE "ID" = TRUE )
GROUP BY YEAR( "Datum" ), MONTH( "Datum" ), "Konto"
ORDER BY "JahrMonat" ASC
```

Die monatliche Übersicht für das Diagramm im Bericht wird aus der "Ansicht_Kasse_mit_Umbuchungen" ermittelt. Im Gegensatz zur Abfrage aus der vorhergehenden Datenbank «Beispiel_Saldo_fortlaufend.odb» wird hier aber wieder der Zeitbereich für die Abfrage über die Filtertabelle begrenzt. Die Abfrage kann bis auf diese Unterabfragen zur Eingrenzug des Datums komplett über die grafische Benutzeroberfläche erstellt werden.

monatlich_Kategorie

```
SELECT YEAR( "Datum" ) || '-' || RIGHT( '0' || MONTH( "Datum" ), 2 )
   AS "JahrMonat", "Kategorie", SUM( "Splitbetrag" ) AS "Summe"
   FROM "Ansicht_Kasse_Kategorie"
   WHERE "Datum"
      BETWEEN ( SELECT "StartDatum" FROM "Filter" WHERE "ID" = TRUE )
      AND ( SELECT "EndDatum" FROM "Filter" WHERE "ID" = TRUE )
GROUP BY YEAR( "Datum" ), MONTH( "Datum" ), "Kategorie"
ORDER BY "JahrMonat" ASC
```

Auch die monatliche Übersicht zu den Kategorien ist nur um die Abfrage der Datumsgrenze nach unten und nach oben erweitert worden. Sie greift ebenfalls auf eine Ansicht zu, allerdings hier der "Ansicht_Kasse_Kategorie".

Konto_Saldo_Konto_Kategorie_Adressat

Die vorherige Abfrage *Konto_Saldo_Konto_Kategorie_Adressat* musste hier lediglich gekürzt werden, da eine Saldenermittlung der Kategorie nicht mehr unbedingt in einem Datensatz der Tabelle "Kasse" möglich ist. Schließlich können einem Datensatz aus "Kasse" gleich mehrere Kategoriebeträge zugeordnet werden.

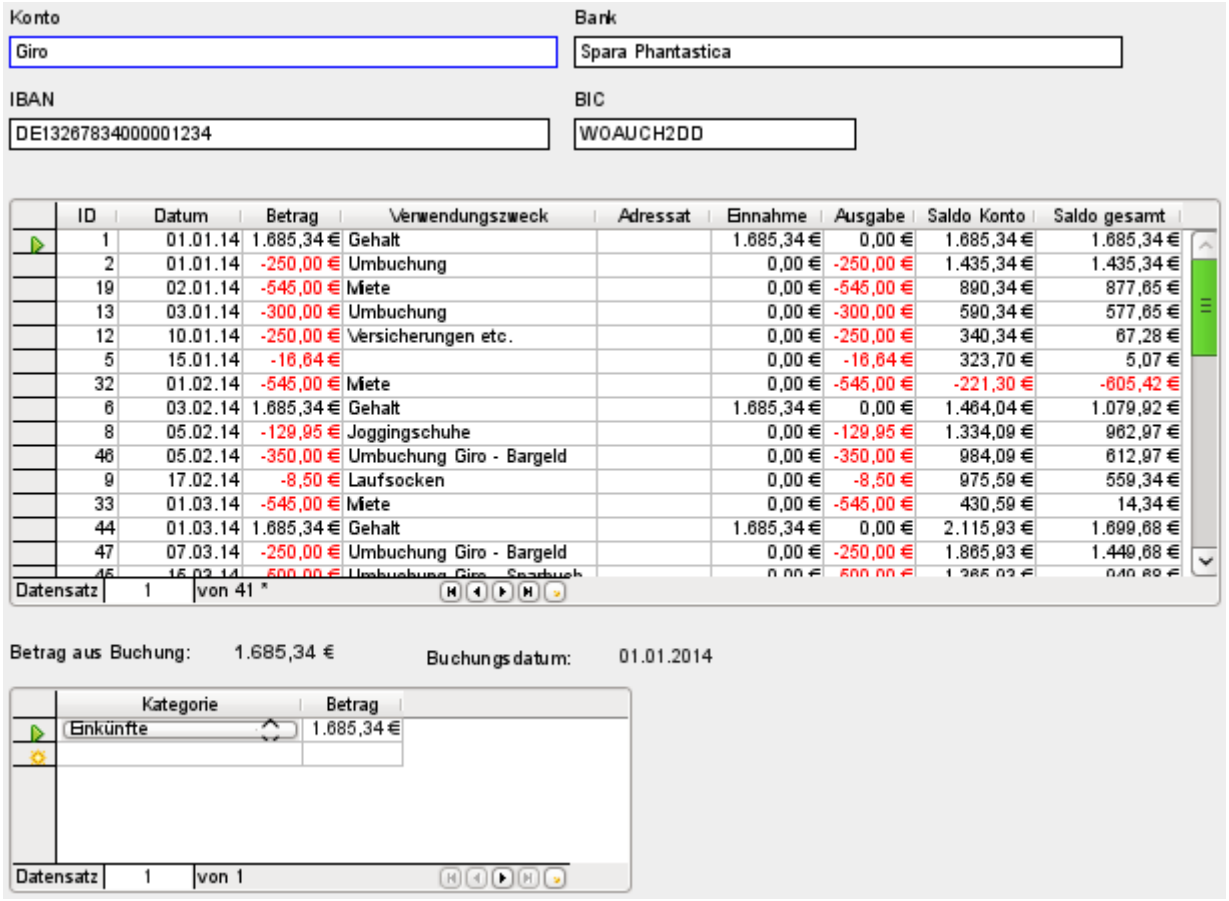
Konto_Saldo_Konto_Kategorie_Adressat_Umbuch

Auch die vorherige Abfrage *Konto_Saldo_Konto_Kategorie_Adressat_Umbuch* musste hier lediglich gekürzt werden, da eine Saldenermittlung der Kategorie nicht mehr unbedingt in einem Datensatz der Tabelle "Kasse" möglich ist.

Formulare

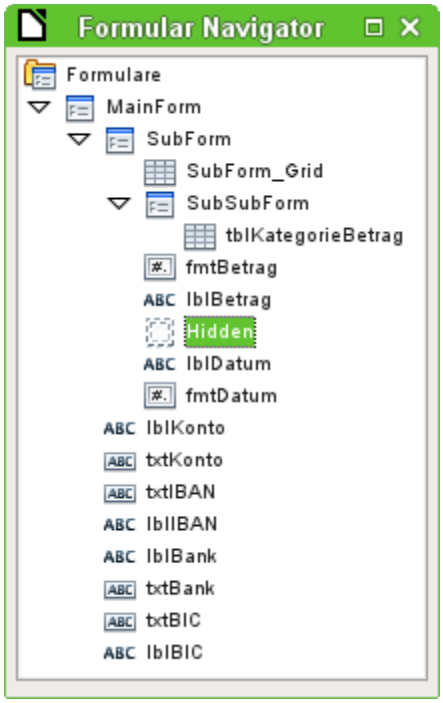
Bis auf ein Formular sind die Formulare vom Aufbau her mit den Formularen der Datenbank «Beispiel_Saldo_fortlaufend.odb» weitgehend identisch. Lediglich die Kategorien müssen in einem Unterformular mit aufgenommen werden, da ja mehrere Kategorien einer Buchung zugewiesen werden können.

Konto



In diesem Formular ist auch eine Neueingabe des Kontos möglich. Im Unterformular dazu liegt ein Tabellenkontrollfeld, das als Datenbasis auf die Abfrage «Konto_Saldo_Konto_Kategorie_Adressat» zugreift. Die Abfrage enthält in dieser Variante keinen Hinweis mehr auf die Kategorie. Stattdessen ist für die Kategorie ein weiteres Unterformular eingerichtet worden, in dem einer Buchung unterschiedliche Kategorien zugewiesen werden können.

Hauptproblem des Formulars ist, dass der Betrag aus der Buchung, der über dem Tabellenkontrollfeld zu Kategorie angezeigt wird, immer gleich der Summe der Beträge sein muss, die im Tabellenkontrollfeld für die Kategorien angezeigt werden. Natürlich lässt sich so etwas durch separate Berechnung erledigen. Besser ist aber, entsprechende Restbeträge auch aufzuzeigen.



Im Formular Navigator ist ein Feld zu sehen, das die Bezeichnung «Hidden» trägt. So ein Feld wird erstellt, indem auf das Formular, hier «SubForm», geklickt wird und mit der rechten Maustaste statt eines neuen Unterformulars ein Verstecktes Steuerelement ausgewählt wird. Solch ein verstecktes Steuerelement hat neben dem Namen (hier: «Hidden») nur noch die Möglichkeit, einen Wert und Zusatzinformationen zu speichern.

Makro «ZeileZwischenspeichern»

Ereignis: **SubForm → Eigenschaften → Ereignisse → Vor der Datensatzaktion**

```
SUB ZeileZwischenspeichern(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oHiddenFeld AS OBJECT
    DIM inUmbuchID AS INTEGER
    DIM stUmbuch AS STRING
    oForm = oEvent.Source
    REM Das Ereignis wird von zwei unterschiedlichen Implementationen ausgelöst.
    REM Nur das Ereignis mit dem ImplementationName
       com.sun.star.comp.forms.ODatabaseForm bietet Zugriff auf die Formularfelder.
    IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
       oHiddenFeld = oForm.getByname("Hidden")
       inUmbuchID = oForm.findColumn("Umbuch_Konto_ID")
       stUmbuch = oForm.getString(inUmbuchID)
       IF oEvent.Action = 3 OR stUmbuch <>"" THEN
       ' Es soll keine Kategoriezuweisung erfolgen, wenn der Datensatz
          gelöscht wird (Action = 3) oder es sich um eine Umbuchung von einem
          auf ein anderes Konto handelt.
          oHiddenFeld.HiddenValue = "no"
       ELSEIF oEvent.Action = 1 THEN
       ' Wenn der Datensatz neu ist (Action = 1), dann existiert noch kein Wert
          für Bookmark. Die Datensatzeingabe erfolgt am Schluss, deswegen
          Anzahl der Zeilen + 1
          oHiddenFeld.HiddenValue = oForm.RowCount + 1
```

```
        ELSE
            oHiddenFeld.HiddenValue = oForm.Bookmark()
        END IF
    END IF
END SUB
```

Die Prozedur «ZeileZwischenspeichern» ist notwendig, damit auf den Datensatz zugegriffen werden kann, für den Eintragungen bei den Kategorien erfolgen sollen. Bei der normalen Bewegung durch ein Formular wird der Speichermodus z.B. durch das Verlassen einer Eingabezeile im Tabellenkontrollfeld ausgelöst. Damit das Unterformular aber mit den korrekten Werten bestückt werden kann muss nach dem Verlassen der Zeile wieder auf die Zeile zurückgesprungen werden. Erst dann sind die Formulare «SubForm» und «SubSubForm» korrekt miteinander verbunden.

Die Prozedur wird durch das Ereignis vor der Datensatzaktion ausgelöst. Mit dem Ereignis sind zwei Implementationen verbunden, so dass das Ereignis jedes normale Makro also doppelt durchlaufen würde. Über den Implementationsnamen wird hier die Implementation heraus gesucht, die einen Zugriff auf die Formularfelder bietet: **`com.sun.star.comp.forms.OdatabaseForm`**.

Das versteckte Feld wird bekannt gemacht. Außerdem wird das Feld herausgesucht, das in der dem Formular zugrundeliegenden Abfrage «Umbuch_Konto_ID» heißt. Dies geht hier über die Funktion **`oForm.findColumn`**. Die Funktion gibt nur eine Zahl wieder, und zwar die Zahl der entsprechenden Tabellenspalte. Über **`oForm.getString`** wird der Wert aus dieser Spalte als Text ausgelesen. Das Auslesen erfolgt hier als Text, da über einen Text leichter leere Felder erkennbar sind.

Wenn das Feld «Umbuch_Konto_ID» nicht leer ist, dann handelt es sich um eine Umbuchung. Folglich wird dort keine Kategoriezuweisung gebraucht. Gleiches gilt, wenn das auslösende Ereignis eine Löschung des Datensatzes bewirken soll. Dies wird über **`oEvent.Action = 3`** abgefragt. Beim Löschen wird schließlich jede Verbindung zu den Kategorien entfernt, eine Eingabe dort ist auch gar nicht mehr möglich. Für diese beiden genannten Fälle wird in dem versteckten Feld die Bezeichnung 'no' gespeichert.

Wenn es sich bei der Eingabe um einen noch nicht existierenden neuen Datensatz handelt (**`oEvent.Action = 1`**), dann kann auf den Datensatz nicht über die Funktion Bookmark zugegriffen werden. Stattdessen wird zu der Anzahl der Zeilen der Tabelle 1 hinzugezählt und der Wert entsprechend in dem versteckten Feld gespeichert. Damit kann anschließend auf die Zeile zurückgesprungen werden.

Handelt es sich um eine Datensatzänderung, dann wird über **`oForm.Bookmark()`** die entsprechende Zeile gespeichert.

Nach Ablauf dieser Prozedur erfolgt die Speicherung des aktuellen Datensatzes in dem Tabellenkontrollfeld. Nach der Datensatzaktion startet das folgende Makro.

Makro «Splitrest»

Mit dieser Prozedur soll berechnet werden, welcher Betrag in der Splittabelle für die Kategorien zur Verfügung steht. Der Betrag soll direkt in den neuen Datensatz eingegeben werden. Es ist also nur noch die Wahl einer Kategorie erforderlich.

Wird der Betrag nur zum Teil einer Kategorie zugeordnet, so wird das Makro aus SubSubForm heraus erneut aufgerufen und dem nächsten Datensatz in der Splittabelle wird der verbleibende Rest zugewiesen – so lange, bis kein Rest mehr bleibt.

Ereignis: **SubForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

und: **SubSubForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

```
SUB Splitrest(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oSubForm AS OBJECT
```

```
DIM oFeld AS OBJECT
DIM oHiddenFeld AS OBJECT
DIM oController AS OBJECT
DIM oView AS OBJECT
DIM oDatenquelle AS OBJECT
DIM oVerbindung AS OBJECT
DIM oSQL_Anweisung AS OBJECT
DIM stSql AS STRING
DIM oSQL_Anweisung2 AS OBJECT
DIM stSql2 AS STRING
DIM oAbfrageergebnis AS OBJECT
DIM inID AS INTEGER
DIM loID AS LONG
DIM inBetrag AS INTEGER
DIM doBetrag AS DOUBLE
DIM doRest AS DOUBLE
oDoc = thisComponent
IF oEvent.Source.hasByName("Hidden") THEN
   oForm = oEvent.Source
ELSE
   oForm = oEvent.Source.Parent
END IF
oHiddenFeld = oForm.getByname("Hidden")
IF oHiddenFeld.HiddenValue = "no" THEN
   EXIT SUB
END IF
```

Da die Prozedur aus Formularen mit unterschiedliche Formularstruktur (Formular, Unterformular usw.) ausgelöst wird muss zuerst geklärt werden, ob das Formular auch das gewünschte Formular ist. In dem Formular soll sich das Feld mit der Bezeichnung «Hidden» befinden.

Ist in dem Feld «Hidden» der **`HiddenValue = "no"`**, so wird die Prozedur nicht weiter ausgeführt. Entweder handelt es sich in diesem Fall um eine Umbuchung oder der auslösende Datensatz aus dem Formular wurde gelöscht. Eine Zuweisung von Kategorien ist also nicht erforderlich oder sogar unmöglich.

```
oDatenquelle = ThisComponent.Parent.CurrentController
If NOT (oDatenquelle.isConnected()) THEN
   oDatenquelle.connect()
END IF
oVerbindung = oDatenquelle.ActiveConnection()
oSQL_Anweisung = oVerbindung.createStatement()
oSQL_Anweisung2 = oVerbindung.createStatement()
IF oHiddenFeld.HiddenValue <> "" THEN
   oForm.absolute(oHiddenFeld.HiddenValue)
   ' Sprung muss nur einmal vollzogen werden. Wert löschen.
   ' Wenn nur in den Kategorieeinweisungen etwas geändert wird, nicht aber
      vorher eine Änderung im Datensatz des Hauptformulars erfolgte,
   ' wird sonst ein Sprung zum zuletzt geänderten Datensatz vollzogen.
   oHiddenFeld.HiddenValue = ""
END IF
```

Der Wert aus dem versteckten Feld wird über oForm.absolute() zum Sprung auf den Datensatz aus dem Eingabeformular für die Buchung genutzt. Das versteckte Feld muss nur einmal ausgelesen werden. Es sollte auf jeden Fall geleert werden, wenn die Prozedur durchlaufen wurde.

Wird grundsätzlich das Makro erst von SubForm ausgelöst und dann von SubSubForm eine Eingabe getätigt, so ist die Zeile auf jeden Fall korrekt. Es könnte aber passieren, dass bei einer Auslösung des Makros direkt aus dem Tabellenkontrollfeld für die Kategoriezuweisung ohne vorherige Änderung des Buchungsdatensatzes ein Sprung zu einem Buchungsdatensatz erfolgt, der zuletzt geändert wurde.

```
inID = oForm.findColumn("ID")
loID = oForm.getLong(inID)
IF oHiddenFeld.Tag <> "" THEN
   stSql = "SELECT ""Kategorie_ID"", ""Betrag"" FROM ""rel_Kasse_Kategorie""
      WHERE ""Kasse_ID"" ='"+oHiddenFeld.Tag+"'"
```

```
            oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
            oHiddenFeld.Tag = ""
            WHILE oAbfrageergebnis.next
                doBetrag = oAbfrageergebnis.getDouble(2)
                stSql2 = "INSERT INTO ""rel_Kasse_Kategorie"" (""Kasse_ID"", "_
                    & """Kategorie_ID"", ""Betrag"") VALUES"_
                    & "('"+loID+"','"+oAbfrageergebnis.getShort(1)+"','"+str(doBetrag)+"')"
                oSQL_Anweisung2.executeUpdate(stSql2)
            WEND
        END IF
```

Zuerst wird nach dem Wert des Primärschlüsselfeldes aus dem Buchungsformular gesucht. Die anschließende Bedingungsabfrage trifft nur zu, wenn es sich um eine periodische Buchung handelt. Diese Erweiterung ist für das Formular *Buchung_Umbuchung_Salden* notwendig.

Bei einer periodischen Buchung soll auch die Kategoriezuweisung automatisch erfolgen. Eine periodische Buchung wird aber nicht vollautomatisch ausgeführt sondern liest nur zuerst einmal die Werte der vorhergehenden Buchung in das Buchungsformular ein. Deshalb kann nicht der komplette Datensatz direkt erstellt werden. Über **`oHiddenFeld.Tag`** wird der Schlüsselwert des Datensatzes ausgelesen, aus dem die periodische Buchung erfolgt ist. Über diesen Schlüsselwert können alle Einträge aus der Tabelle «rel_Kasse_Kategorie» ausgelesen werden, die zu dem Buchungsdatensatz passen. Sie werden nacheinander als neue Datensätze mit der aktuellen Schlüsselnummer **`loID`** in die Tabelle «rel_Kasse_Kategorie» neu eingefügt. Dabei erweist sich das Einfügen des Betrages als etwas problematisch. Wird der Wert nicht in einen String umgewandelt (**`str(doBetrag)`**), so gibt Basic statt eines Wertes mit Dezimalpunkt einen Wert mit der Formatierung des eingestellten Landes der GUI wieder, also in diesem Fall ein Dezimalkomma. Das führt unweigerlich dazu, dass der Datensatz nicht eingefügt wird.

```
        stSql = "SELECT ""a"".""Betrag"" - IFNULL((SELECT SUM(""Betrag"") "_
            & "FROM ""rel_Kasse_Kategorie"" WHERE ""Kasse_ID"" = ""a"".""ID""),0.00) "_
            & "AS ""Wert"" FROM ""Kasse"" AS ""a"" WHERE ""a"".""ID"" = '"+loID+"'"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            doRest = oAbfrageergebnis.getDouble(1)
        WEND
        oSubForm = oForm.getByname("SubSubForm")
        inBetrag = oSubForm.findColumn("Betrag")
        oFeld = oSubForm.getByName("tblKategorieBetrag")
        oController = oDoc.getCurrentController()
        oView = oController.getControl(oFeld)
        oView.setFocus
        wait 500  ' kurze Pause, da sonst der Zugriff auf das Unterformular nicht
            funktioniert
        IF doRest <> 0 THEN
            oSubForm.MoveToInsertRow
            oSubForm.UpdateDouble(inBetrag,doRest)
        ELSE
            EXIT SUB
        END IF
    END SUB
```

Anschließend wird über eine Abfrage festgestellt, wie hoch der Betrag zu der passenden Buchungsnummer abzüglich der Summe der Beträge zu der entsprechenden Buchungsnummer in der Tabelle «rel_Kasse_Kategorie» ist. Danach wird das Unterformular bestimmt und der Cursor in das Tabellenkontrollfeld gesetzt.

In den Test hat es sich als notwendig erwiesen, vor weiteren Aktionen mit dem Unterformular für die Kategorieeingabe eine kurze Wartezeit einzulegen. Das Makro gelang mit einer Messagebox-Nachfrage; sobald aber die Meldung vom Bildschirm genommen wurde konnte kein Betrag an das feld im Tabellenkontrollfeld übergeben werden.

Ist nach der Abfrage noch ein Rest zu verzeichnen, so wird eine neue Zeile in dem Unterformular «SubSubForm» erstellt. In diese Zeile wird der Restbetrag eingetragen. Ist hingegen kein Rest mehr vorhanden, so erübrigt sich ein Eintrag. Die Prozedur wird beendet.

Kasse

Das Formular «Kasse» bietet statt der Konstruktion mit Hauptformular und Unterformular ein Tabellenkontrollfeld, in dem das Konto für jeden Datensatz gewählt werden kann. Der Ansatz entspricht dem aus *Kasse* im vorhergehenden Formular.

Die Eingabe der Kategorien erfolgt über das entsprechende separate Unterformular in ein Tabellenkontrollfeld. Die dafür notwendigen Makros stehen im Formular *Konto*.

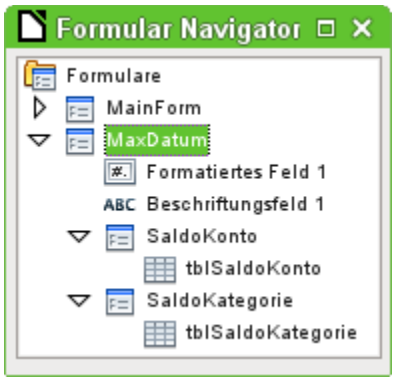
Konto_Salden

Das Formular «Konto_Salden» zeigt neben der Eingabemöglichkeit aus dem Formular «Konto» noch den zum Zeitpunkt des Buchungsdatensatzes aktuellen Kontostand in den verschiedenen Konten und Kategorien an. Allerdings wird, wie bei der vorhergehenden Datenbank, auch hier kein Kontostand angezeigt, während ein neuer Datensatz eingegeben wird.

Die Eingabe der Kategorien erfolgt über das entsprechende separate Unterformular in ein Tabellenkontrollfeld. Die dafür notwendigen Makros stehen im Formular *Konto*.

Konto_Salden_komplett

Im Gegensatz zum Formular «Konto_Salden» zeigt dieses Formular immer den Kontostand auf, der aus allen Eingaben in der Tabelle «Kasse» resultiert. Dazu ist es notwendig, dass die Anzeige des Kontostandes in einem separaten Formular neben dem bisherigen Hauptformular «MainForm» erfolgt.



Die Eingabe der Kategorien erfolgt über das entsprechende separate Unterformular in ein Tabellenkontrollfeld. Das *Makro «ZeileZwischenspeichern»* und das *Makro «Splitrest»* wurde bereits im ersten Formular vorgestellt. Das Makro «Splitrest» wird über das Makro «Aktualisieren» gestartet. Das Makro «ZeileZwischenspeichern» wird über **SubForm → Eigenschaften → Ereignisse → Vor der Datensatzaktion** gestartet.

Makro «Aktualisieren»

Ereignis: **SubForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

und: **SubSubForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

```
SUB Aktualisieren(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    Splitrest(oEvent)
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MaxDatum")
    oForm.reload
END SUB
```

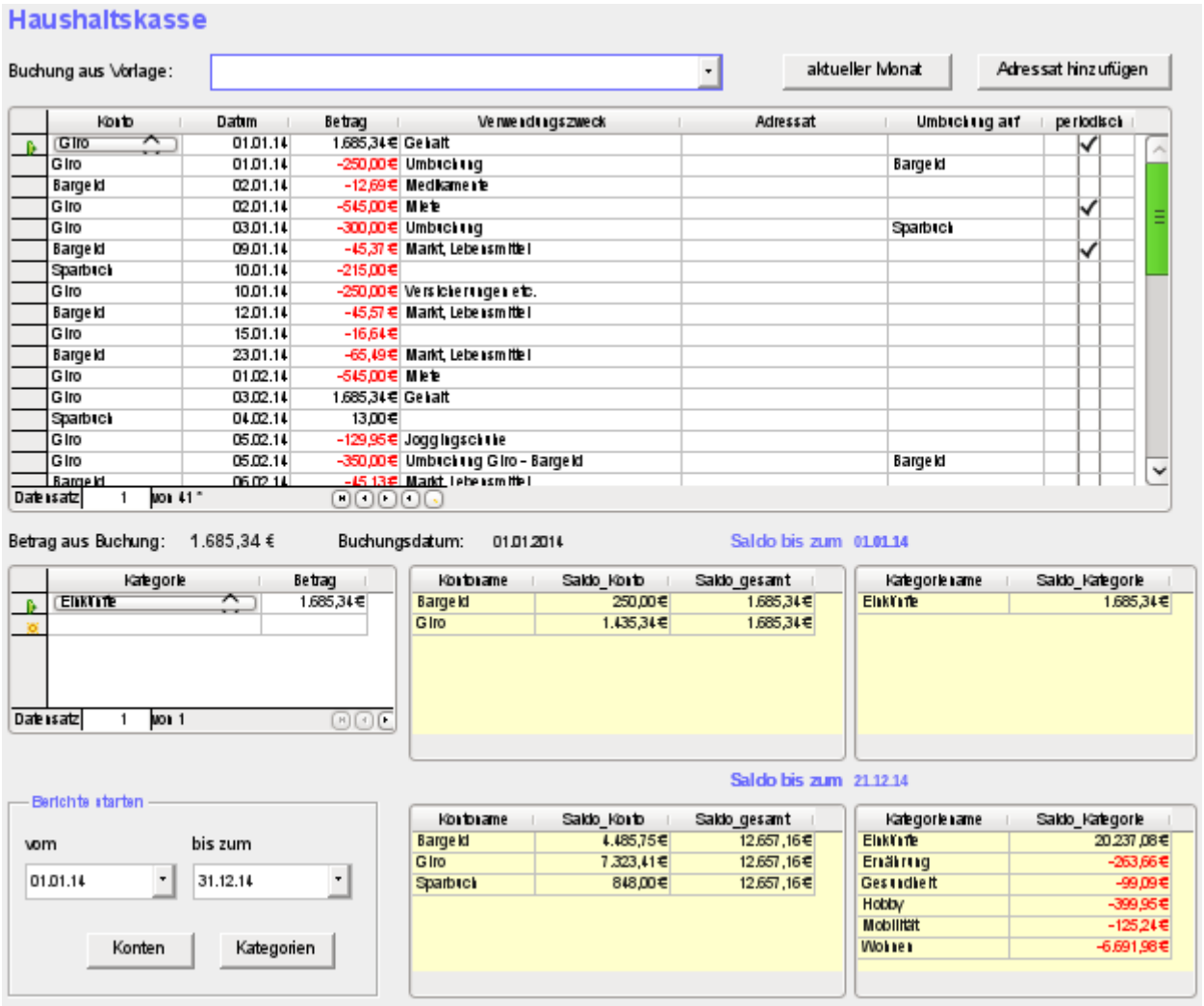
Dieses Makro ist lediglich deshalb erforderlich, weil das Nebenformular sonst von einer Änderung der Daten im Hauptformular nichts mit bekommt. Zuerst wird die eventuelle Splitbuchung nach einer Datensatzänderung durchgeführt. Anschließend wird dann des Nebenformular mit den aktuellen Daten neu eingelesen.

Konto_Salden_komplett_Umbuchung

Neben dem vorhergehenden Formular ist hier noch zusätzlich die Umbuchungsfunktion mit eingebaut. Ansonsten entspricht das Formular dem Formular «Konto_Salden_komplett». Die Makros entsprechen denen im Formular *Konto* und *Konto_Salden_komplett*.

Buchung_Umbuchung_Salden

Dieses Formular soll möglichst viele Vorgaben für eine Verwaltung der Haushaltskasse erfüllen. Es sprengt daher von der Komplexität der Makros auch teilweise den Rahmen eines auf eine besondere Funktion beschränkten Beispiels.

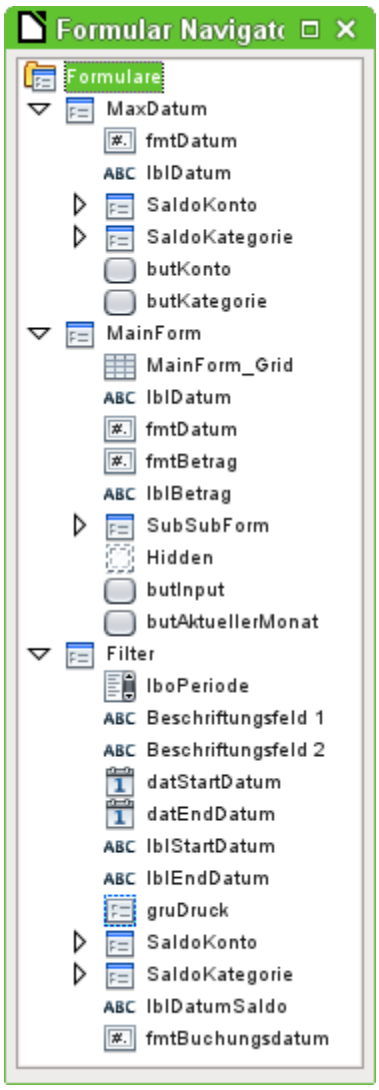


Die Funktionen im Einzelnen, von der Formularansicht her von oben nach unten beschrieben:

- Eine periodische Buchung kann als «Buchung aus Vorlage» ausgesucht werden. Damit werden Buchungsvorgänge, die sich häufig wiederholen (z.B. 'Miete', 'Gehalt') einfach in das Formular übernommen. Es muss im Hauptformular lediglich das Datum ergänzt werden. Änderungen sind dort natürlich auch noch möglich.

- Die anzuzeigenden Datensätze können auf den aktuellen Monat durch einen gesonderten Filter beschränkt werden. Ein Runterscrollen bei zu vielen Datensätzen ist so nicht mehr nötig.
- Fehlt ein Adressat in der Auswahlliste, so kann er über ein einfaches Eingabefeld neu eingegeben werden. Er steht dann anschließend in der Auswahl zur Verfügung.
- Das Hauptformular zeigt neben den bisherigen Felder noch die Möglichkeit an, Datensätze zu periodischen Datensätzen zu erklären, die dann in der oben genannten Listenfeldauswahl erscheinen.
- Die Kategorieeingabe unterscheidet sich nicht von der der vorhergehenden Formulare. Neben der Kategorieeingabe existiert allerdings die Übersicht über den Kontostand zum Zeitpunkt der Buchung. Im Gegensatz zu den vorherigen Lösungen wird hier auch ein Wert angezeigt, wenn ein neuer Datensatz eingegeben wird. In dem Falle springt die Anzeige auf den Datenstand, der mit dem aktuellen Eingabedatum zusammen hängt.
- Die Berichte zum Verlauf der Buchungen bezogen auf die Konten und Kategorien können, beschränkt auf eine vorher erstellte Datumsauswahl, direkt aus dem Formular heraus gestartet werden.
- Neben dem Berichtsstart wird dann der Kontostand angezeigt, der dem höchsten Buchungsdatum entspricht. Liegt so ein Buchungsdatum in der Zukunft, so kann die Anzeige hier anders ein als die Anzeige, die bei einer Neueingabe in den darüber liegenden Tabellenkontrollfeldern erfolgt.

Der Formularnavigator zeigt für dieses Formular gleich drei nebeneinander liegende Formulare: «MaxDatum», «MainForm» und «Filter». Die Filtertabelle wird für die periodischen Buchungen, die Datumsbegrenzung der Berichte und die Anzeige von momentanem Buchungsstand bei der Eingabe neuer Datensätze benötigt. Entsprechend enthält das dazugehörige Formular «Filter» auch die zugehörigen Eingabefelder und als Unterformulare die Tabellenkontrollfelder, die den momentanen Kontostand in Abhängigkeit vom Datensatz mitteilen sollen.



Die folgenden Makros werden benötigt:

Ereignis: **MainForm → Eigenschaften → Ereignisse → Vor der Datensatzaktion**

startet das *Makro «ZeileZwischenspeichern»*

Ereignis: **MainForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

und: **SubSubForm → Eigenschaften → Ereignisse → Nach der Datensatzaktion**

startet das *Makro «Aktualisieren»*

Makro «Aktuelles_Buchungsdatum»

Das sorgt dafür, dass die Salden für das Konto und die Kategorie abhängig von dem aktuellen Datensatz ermittelt werden können und, wenn gerade eine Neueingabe gemacht wird, abhängig vom aktuellen Datum dargestellt werden.

Ereignis: **MainForm → Eigenschaften → Ereignisse → Nach dem Datensatzwechsel**

```
SUB Aktuelles_Buchungsdatum(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSql AS STRING
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm2 AS OBJECT
```

```
    oForm = oEvent.Source
    IF oForm.IsNew THEN
       stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = NOW() WHERE ""ID"" = TRUE"
    ELSE
       stSql = "UPDATE ""Filter"" SET ""DatensatzDatum"" = '" +
          oForm.getString(oForm.findColumn("Datum")) + "' WHERE ""ID"" = TRUE"
    END IF
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    oSQL_Anweisung.executeUpdate(stSql)
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm2 = oDrawpage.forms.getByName("Filter")
    oForm2.reload
END SUB
```

Nach dem Datensatzwechsel im Hauptformular wird ermittelt, ob ein neuer Datensatz eingefügt werden soll: **`oForm.IsNew`**. Handelt es sich um einen neuen Datensatz, so wird in der Tabelle "Filter" das Feld "DatensatzDatum" auf das aktuelle Datum **`NOW()`** eingestellt. Handelt es sich nicht um ein einen neuen Datensatz, so wird stattdessen in der Tabelle "Filter" das Feld "DatensatzDatum" auf den Wert des aktuellen Datums des gerade erreichten Datensatzes eingestellt: **`oForm.getString(oForm.findColumn("Datum"))`**.

Der SQL-Befehl wird an die Datenbank verschickt und das Nebenformular «Filter» auf den aktuellen Wert entsprechend eingestellt.

Makro «periodische_Buchung»

Im Hauptformular können Datensätze als periodische Datensätze gekennzeichnet werden. Diese Datensätze werden dann als Vorlage für andere Datensätze über das Listenfeld ganz oben im Formular abrufbar. Der Inhalt dieser Datensätze wird, bis auf das Buchungsdatum, als neuer Datensatz eingefügt und muss nur noch abgespeichert werden.

Ereignis: **Filter → lboPeriode → Eigenschaften → Ereignisse → Modifiziert**

```
SUB periodische_Buchung(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oHiddenFeld AS OBJECT
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSql AS STRING
    DIM oAbfrageergebnis AS OBJECT
    DIM loID AS LONG
    oFeld = oEvent.Source.Model
    IF oFeld.CurrentValue = "" THEN
       EXIT SUB
    END IF
    loID = oFeld.CurrentValue
```

Der Wert des Listenfeldes wird ermittelt. Es handelt sich dabei nicht um den angezeigten Wert, sondern um den Inhalt des Schlüsselfeldes, das damit verbunden ist. In diesem Fall ist das der Primärschlüsselinhalt der Tabelle «Kasse».

Dieser Wert wird zur weiteren Nutzung in dem bereits im *Makro «Splitrest»* vorgestellten versteckten Feld, in diesem Fall in den Zusatzinformationen (Tag), zwischengelagert. Er wird dort später für die Splitbuchungen bei den Kategorien benötigt.

```
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
```

```
    oForm = oDrawpage.forms.getByName("MainForm")
    oForm.moveToInsertRow
    oHiddenFeld = oForm.getByName("Hidden")
    oHiddenFeld.Tag = loID
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT ""Betrag"", ""Konto_ID"", ""Adressat_ID"", ""Verwendungszweck"",
       ""Umbuch_Konto_ID"" FROM ""Kasse"" WHERE ""ID"" ='"+loID+"'"
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Der Datensatz zu dem entsprechenden Primärschlüssel aus der Tabelle "Kasse" wird ausgelesen. Die Felder "ID" und "Datum" werden nicht benötigt. "Datum" wird sowieso neu eingestellt und "ID" ist bekannt – außerdem soll ja ein neuer Datensatz erstellt wird, so dass "ID" über den Autowert neu festgelegt wird.

```
    WHILE oAbfrageergebnis.next
       oForm.UpdateDouble(oForm.findColumn("Betrag"),oAbfrageergebnis.getDouble(1))
       oForm.UpdateLong(oForm.findColumn("Konto_ID"),oAbfrageergebnis.getLong(2))
       oForm.UpdateLong(oForm.findColumn("Adressat_ID"),oAbfrageergebnis.getLong(3))
       IF oAbfrageergebnis.wasNull() THEN
```

Mit **`wasNull()`** wird das letzte Ergebnis jeweils getestet. Damit wird ausgeschlossen, dass stattdessen '0' für die "Adressat_ID" weiter gegeben wird, obwohl dort vielleicht kein Wert enthalten ist. "Betrag" und "Konto_ID" hingegen sind Felder, die nach Tabellendefinition nie **`NULL`** sein dürfen.

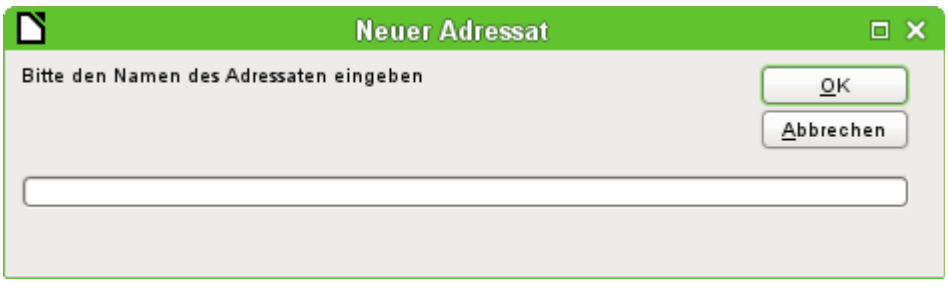
Der folgende "Verwendungszweck" darf zwar auch **`NULL`** sein. Wenn dort allerdings ein leerer String eingetragen wird, so schadet dies nicht der Integrität der Dateneingabe. Bei der "Umbuch_Konto_ID" muss hingegen wieder darauf geachtet werden, dass tatsächlich **`NULL`** übermittelt wird.

```
          oForm.UpdateNULL(oForm.findColumn("Adressat_ID"),NULL)
       END IF
       oForm.UpdateString(oForm.findColumn("Verwendungszweck"),
          oAbfrageergebnis.getString(4))
       oForm.UpdateLong(oForm.findColumn("Umbuch_Konto_ID"),
          oAbfrageergebnis.getLong(5))
       IF oAbfrageergebnis.wasNull() THEN
          oForm.UpdateNULL(oForm.findColumn("Umbuch_Konto_ID"),NULL)
       END IF
    WEND
    oSQL_Anweisung.executeUpdate("UPDATE ""Filter"" SET ""Kasse_ID"" = NULL WHERE
       ""ID"" = TRUE")
    oFeld.Parent.reload
END SUB
```

Zum Schluss wird der Wert des Listenfeldes über den entsprechenden Wert in der Tabelle "Filter" wieder zurück gesetzt.

Makro «Adressat_neu»

Am Anfang fehlen noch viele Adressaten. Hier soll auf einfach Weise wenigstens ein Name für den Adressaten abgefragt werden, damit eine Eingabe zügig erfolgen kann.



Solch ein Eingabefeld wird im Folgenden gebildet. Der Inhalt wird eingelesen, auf mögliche gleiche Inhalte hin überprüft und ansonsten in die Datenbank übertragen.

Ereignis: **MainForm → butInput → Eigenschaften → Ereignisse → Aktion ausführen**

```
SUB Adressat_neu(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM oListbox AS OBJECT
    DIM stInput AS STRING
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oSQL_Anweisung2 AS OBJECT
    DIM stSql AS STRING
    DIM oAbfrageergebnis AS OBJECT
    stInput = InputBox("Bitte den Namen des Adressaten eingeben","Neuer Adressat")
    stInput = Trim(stInput)
    IF stInput = "" THEN
        msgbox "Der Eintrag für den Adressaten ist leer."
        EXIT SUB
    END IF
```

Der Inhalt der **`InputBox`** wird als Text in **`stInput`** gespeichert. Um unnötige Leertasten auszuschließen wird **`stInput`** mit **`Trim(stInput)`** von Leertasten vor und nach der sichtbaren Eingabe befreit.

Ist der Inhalt leer, so wird das Makro nach einer kurzen Meldung beendet. Ansonsten wird eine Verbindung zur Datenquelle aufgebaut, sofern noch nicht vorhanden. Da eine SQL-Anweisung abgesetzt wird, während die andere noch im Speicher liegt, werden für die SQL-Anweisungen zwei Objekte erstellt.

In der Datenbank wird jetzt nachgefragt, ob der "Name" in genau der Schreibweise schon vorhanden ist. Bei dieser Nachfrage wird nach Groß- und Kleinschreibung unterschieden. Wenn klar ist, dass der "Name" nicht bereits vor kommt wird die zweite Anweisung, nämlich das Einfügen des neuen Datensatzes, abgearbeitet.

```
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    oSQL_Anweisung2 = oVerbindung.createStatement()
    stSql = "SELECT ""Name"" FROM ""Adressat"" WHERE ""Name"" ='"+stInput+"'"
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    IF oAbfrageergebnis.Next THEN
        msgbox "Der Adressat """+stInput+""" ist schon vorhanden."
    ELSE
        oSQL_Anweisung2.executeUpdate("INSERT INTO ""Adressat"" (""Name"") VALUES ('" +
            stInput + "')")
        oForm = oEvent.Source.Model.Parent
        oListbox = oForm.getByName("MainForm_Grid").getByName("lboAdressat")
        oListbox.refresh
    END IF
END SUB
```

Nachdem ein neuer Datensatz eingefügt wurde ist es erforderlich, dass das Listenfeld seinen Datenbestand neu einliest. Da der Button, der das Makro ausgelöst hat, genau im gleichen Formular liegt wie das Tabellenkontrollfeld für die Buchungseingabe, kann das Formular über **`oEvent.Source.Model.Parent`** erreicht werden. Das Listenfeld liegt im Tabellenkontrollfeld. Das Tabellenkontrollfeld hat den Namen «MainForm_Grid», das Listenfeld den Namen «lboAdressat». Über **`oListbox.refresh`** wird der Inhalt des Listenfeldes erneuert.

Makro «Monat_aktuell»

Ereignis: **MainForm → butAktuellerMonat → Eigenschaften → Ereignisse → Aktion ausführen**

```
SUB Monat_aktuell(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    oForm = oEvent.Source.Model.Parent
    oForm.Filter = "YEAR(""Datum"") = YEAR(NOW()) AND MONTH(""Datum"") = MONTH(NOW())"
    oForm.ApplyFilter = True
    oForm.reload
END SUB
```

Der Button für den Filter liegt in dem Formular, dessen Filter entsprechend eingestellt werden soll. Der Filtertext enthält den SQL-Befehl, mit dem auch eine Tabelle entsprechend gefiltert würde:

```
SELECT * FROM "Kasse" WHERE YEAR("Datum") = YEAR(NOW()) AND
MONTH("Datum") = MONTH(NOW())
```

Der Filter wird mit **`oForm.Filter`** zugewiesen. Mit **`oForm.ApplyFilter`** wird der Filter eingestellt. Er kann also auch später über die Navigationsleiste wieder aus- und erneut eingestellt werden. Zum Schluss wird das Formular neu eingelesen.

Makro «Bericht_starten»

Ereignis: **MaxDatum → butKonto → Eigenschaften → Ereignisse → Aktion ausführen**

und: **MaxDatum → butKategorie → Eigenschaften → Ereignisse → Aktion ausführen**

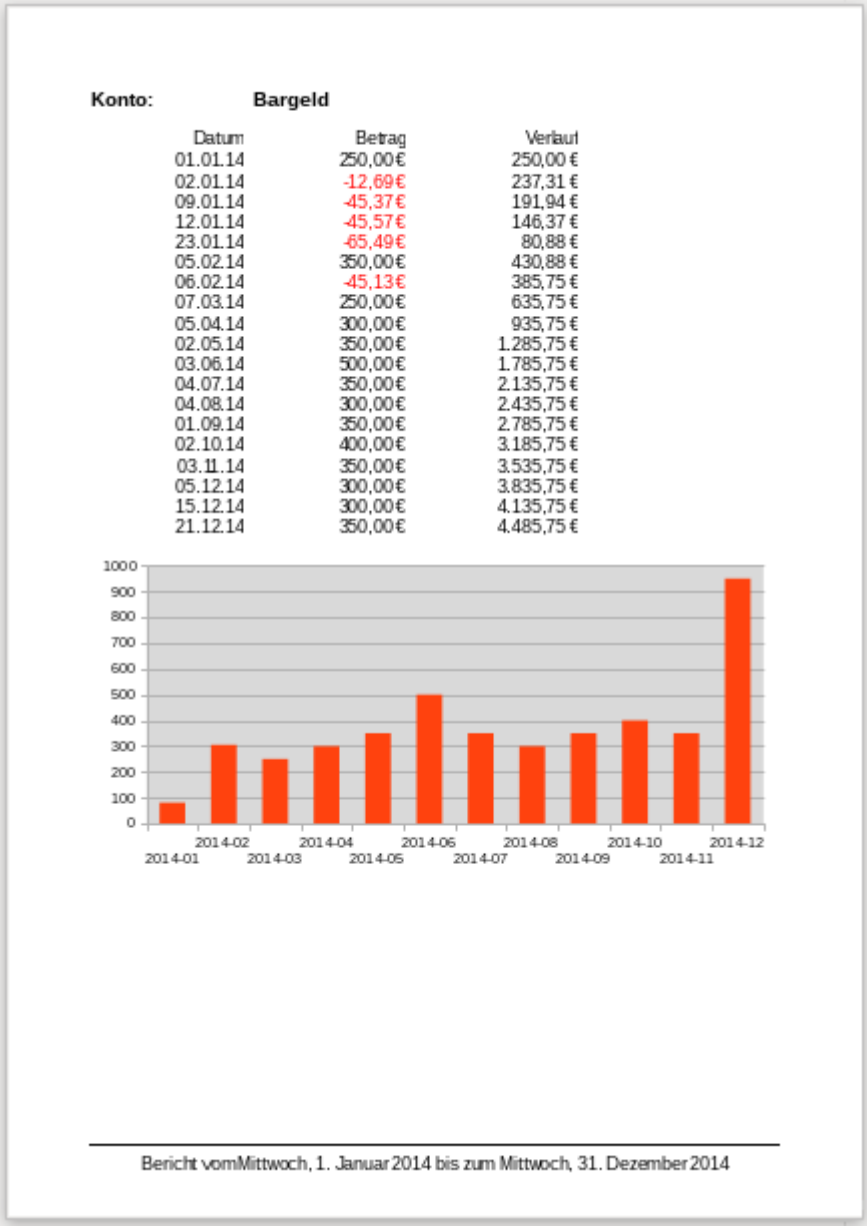
```
SUB Bericht_starten(oEvent AS OBJECT)
    DIM stBericht AS STRING
    stBericht = oEvent.Source.Model.Tag
    ThisDatabaseDocument.ReportDocuments.getByName(stBericht).open
END SUB
```

Die beiden Buttons haben in ihren Zusatzeigenschaften den Namen des Berichtes stehen, den sie öffnen sollen. Für das Öffnen des Berichtes reicht dann einfach der obige Befehl mit **`.open`**.

Berichte

Die Datenbank hat zwei Berichte, die vom Prinzip her gleich konstruiert sind wie die *Berichte* aus dem vorhergehenden Formular.

Einziger Unterschied zu den Berichten: Die Ausgabe der berichte ist über die Abfrage an eine bestimmte Zeitspanne gekoppelt. Daher erscheint auch, wie der folgende Screenshot zeigt, in der Fußleiste eine Zeitspanne, die den Zeitraum aufzeigt, für den der Bericht erstellt wurde.



Konto: **Bargeld**

Datum	Betrag	Verlauf
01.01.14	250,00€	250,00 €
02.01.14	-12,69€	237,31 €
09.01.14	-45,37€	191,94 €
12.01.14	-45,57€	146,37 €
23.01.14	-65,49€	80,88 €
05.02.14	350,00€	430,88 €
06.02.14	-45,13€	385,75 €
07.03.14	250,00€	635,75 €
05.04.14	300,00€	935,75 €
02.05.14	350,00€	1.285,75 €
03.06.14	500,00€	1.785,75 €
04.07.14	350,00€	2.135,75 €
04.08.14	300,00€	2.435,75 €
01.09.14	350,00€	2.785,75 €
02.10.14	400,00€	3.185,75 €
03.11.14	350,00€	3.535,75 €
05.12.14	300,00€	3.835,75 €
15.12.14	300,00€	4.135,75 €
21.12.14	350,00€	4.485,75 €

Bericht vomMittwoch, 1. Januar 2014 bis zum Mittwoch, 31. Dezember 2014

Autotext, Markierung von Suchergebnissen und Rechtschreibprüfung

Das «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb» ist in Teilen auch im Handbuch erklärt. Erleichterungen aus der Nutzung des Writer werden hier teilweise auch in Base verfügbar gemacht. So entsteht beim Autotext nach der Eingabe von MFG 'Mit freundlichen Grüßen …'. Bei der Rechtschreibung werden falsch geschriebene Worte mit einer roten Schlängellinie versehen. Die Suchmarkierung schließlich zeigt vor allem bei einem größeren Textanteil auf, an welcher Stelle der Suchbegriff gefunden wurde.

Tabellen

Die Datenbank besteht im sparsamsten Aufbau aus zwei Tabellen.

In der Tabelle "Tabelle" werden folgende Felder definiert:

Feldname	Feldtyp	Beschreibung
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann auch als Auto-Wert-Feld gesetzt werden.
Memo	Memo	Hier kann sehr viel Text eingegeben werden. Das Feld würde auch ganze Bücher fassen. Allerdings ist der Text nicht so formatiert, dass er auch entsprechend gut gegliedert wäre. Über ein Textfeld kann hier standardmäßig neben den üblichen Zeichen nur der Zeilenumbruch eingegeben werden.

In der Tabelle "Filter" werden folgende Felder definiert:

Feldname	Feldtyp	Beschreibung
ID	Ja/Nein	Primärschlüssel der Tabelle. Diese Tabelle nimmt nur einen Datensatz auf.
Suchtext	Text	Hier wird der Suchtext zwischengespeichert, der in den Formularfeldern angezeigt werden soll und auf den die Suchabfrage zugreift.

Die Tabellen werden nicht miteinander verknüpft. Schließlich soll nur demonstriert werden, wie eine entsprechende Anzeige in einem Formularfeld machbar ist.

Abfragen

Die Abfrage «Abfrage_Formular_Like» dient dazu, die Datensätze entsprechend auf das Suchergebnis einzuschränken. Für die Formatierung der Suche ist sie ohne Bedeutung.

```
SELECT "ID", "Memo"
FROM "Tabelle"
WHERE
    LOWER ( "Memo" )
        LIKE '%' ||
        LOWER ( IFNULL( ( SELECT "Suchtext" FROM "Filter" WHERE "ID" = TRUE ), '' ) )
        || '%'
```

Aus der Tabelle "Filter" wird der Eintrag in dem Feld "Suchtext" ausgelesen. Der Eintrag wird in Kleinbuchstaben umgewandelt und mit dem, ebenfalls in Kleinbuchstaben umgewandelten, Eintrag im Feld "Memo" verglichen. Taucht der Begriff irgendwo in dem Feld auf, so wird er entsprechend angezeigt.

Bei Feldern mit recht großem Inhalt kann es nützlich sein, direkt die Position der entsprechenden Fundstelle angegeben zu bekommen. Dafür sind mehrere gestaffelte Abfragen aufgelistet, die den Suchbegriff mit Position wieder geben. Eine ausführliche Erklärung zu diesen Abfragen ist im Handbuch enthalten. Die oben genannte Abfrage könnte eventuell über die Suche der Position noch beschleunigt werden. Hier die Abfrage «Abfrage_Formular_Locate», die die vorhergehende Abfrage ersetzen könnte:

```
SELECT "ID", "Memo"
FROM "Tabelle"
WHERE
    LOCATE(
        LOWER ( IFNULL( ( SELECT "Suchtext" FROM "Filter" WHERE "ID" = TRUE ), '' ) ),
        LOWER ( "Memo" ) )
        > 0
```

Die Abfrage sucht über LOCATE nach dem ersten Treffer. Wenn ein Treffer enthalten ist, dann wird die Position wieder gegeben und die Suche nicht weiter fortgeführt.

Formulare

Die Formulare benötigen grundsätzlich den Zugriff auf Makros. Am einfachsten aufgebaut ist noch das Formular «Autotext», das direkt nur mit Feldern für die Tabelle "Tabelle" arbeitet.

Autotext

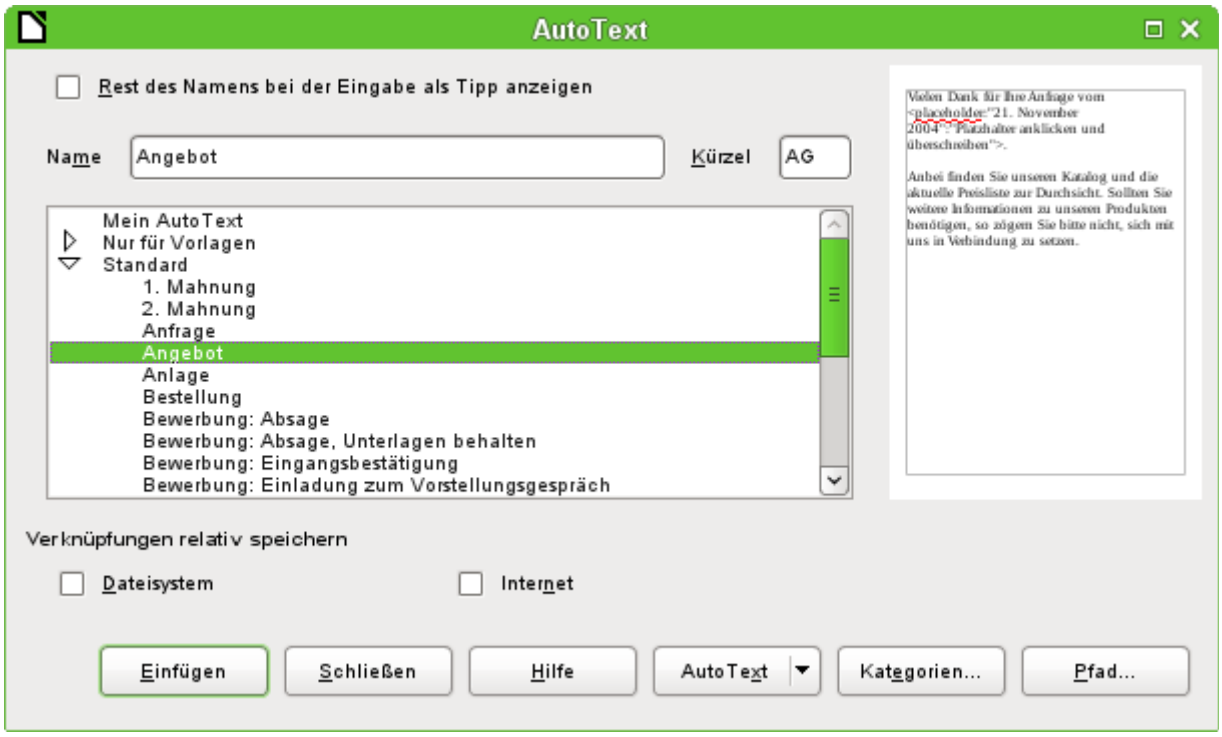
In diesem Formular gibt es lediglich ein Eingabefeld für den Primärschlüssel sowie ein (groß gezogenes) Eingabefeld für die mehrzeilige Eingabe. Hier ist **Eigenschaften → Allgemein → Text-Typ → Mehrzeilig** eingestellt. Das Feld ist nicht formatierbar, da diese Felder nicht direkt mit einem Datenfeld der Tabelle verknüpft werden können.

Über **Eigenschaften → Ereignisse → Text modifiziert** wird von dem Memo-Feld das Makro «Autotext» gestartet.

```
SUB Autotext(oEvent AS OBJECT)
    DIM arText()
    DIM stText AS STRING
    DIM oSelection AS NEW com.sun.star.awt.Selection
    DIM oFeld AS OBJECT
    DIM oAutotexte AS OBJECT
    DIM oBereich AS OBJECT
    DIM oCursor AS OBJECT
    oFeld = oEvent.Source
    IF oFeld.text <> "" THEN
        arText = Split(oFeld.text)
        stText = arText(Ubound(arText))
        oAutotexte = createunoservice("com.sun.star.text.AutoTextContainer")
        oBereich = oAutotexte.getByName("standard")
```

Autotext wird im Writer in den folgenden Bereichen abgespeichert:

`template` «Nur für Vorlagen»
`standard` «Standard»
`crdbus` «Visitenkarten, gesch. (85x50)»
`mytexts` «Mein AutoText»



Autotext wird im Writer mit den festgelegten Zeichen und F3 gestartet. Die Einträge können über **Bearbeiten → AutoText** im Writer bearbeitet werden.

```
        IF oBereich.hasByName(stText) THEN
            oAutotext = oBereich.getByName(stText)
```

In dem Bereich wird nachgesehen, ob das getippte Kürzel vielleicht enthalten ist. So ergibt das in der oben gezeigten Abbildung eingegebene 'AG' den Autotext für das «Angebot». Solange die Autotextinhalte nicht allzu umfangreich sind («Standard» hat beim Start 20 Einträge) kann diese Suche nach jeder Buchstabeneingabe problemlos erfolgen. Bei umfangreicheren Listen sollte vielleicht in Betracht gezogen werden, dass der Autotext erst nach der Eingabe einer Leertaste startet. Die Variable **`stText`** wurde zu Beginn des Makros dadurch gebildet, dass von dem in dem Eingabefeld enthaltenen Text einfach der Text extrahiert wurde, der nach dem letzten Leerzeichen erschien.

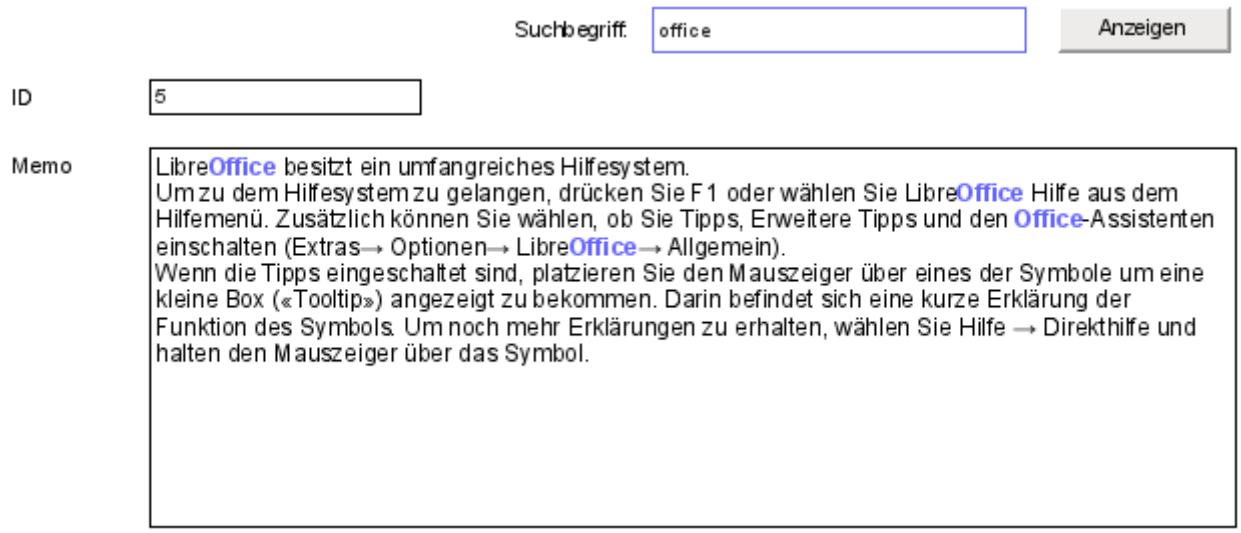
```
            oCursor = oFeld.Model.createTextCursor()
            oCursor.gotoEnd(true)
            oCursor.goLeft(len(stText),true)
            oFeld.Model.insertString(oCursor,oAutotext.String,true)
```

In dem Feld wird ein Textcursor erstellt. Mit dem Cursor wird das Kürzel für den Autotext markiert. Der zugehörige Text für die Abkürzung wird ausgelesen und an der Position des Cursors eingefügt. Anschließend wird dafür gesorgt, dass der sichtbare Cursor in dem Textfeld an das Ende des Textes rutscht und keine Markierung hinterlässt.

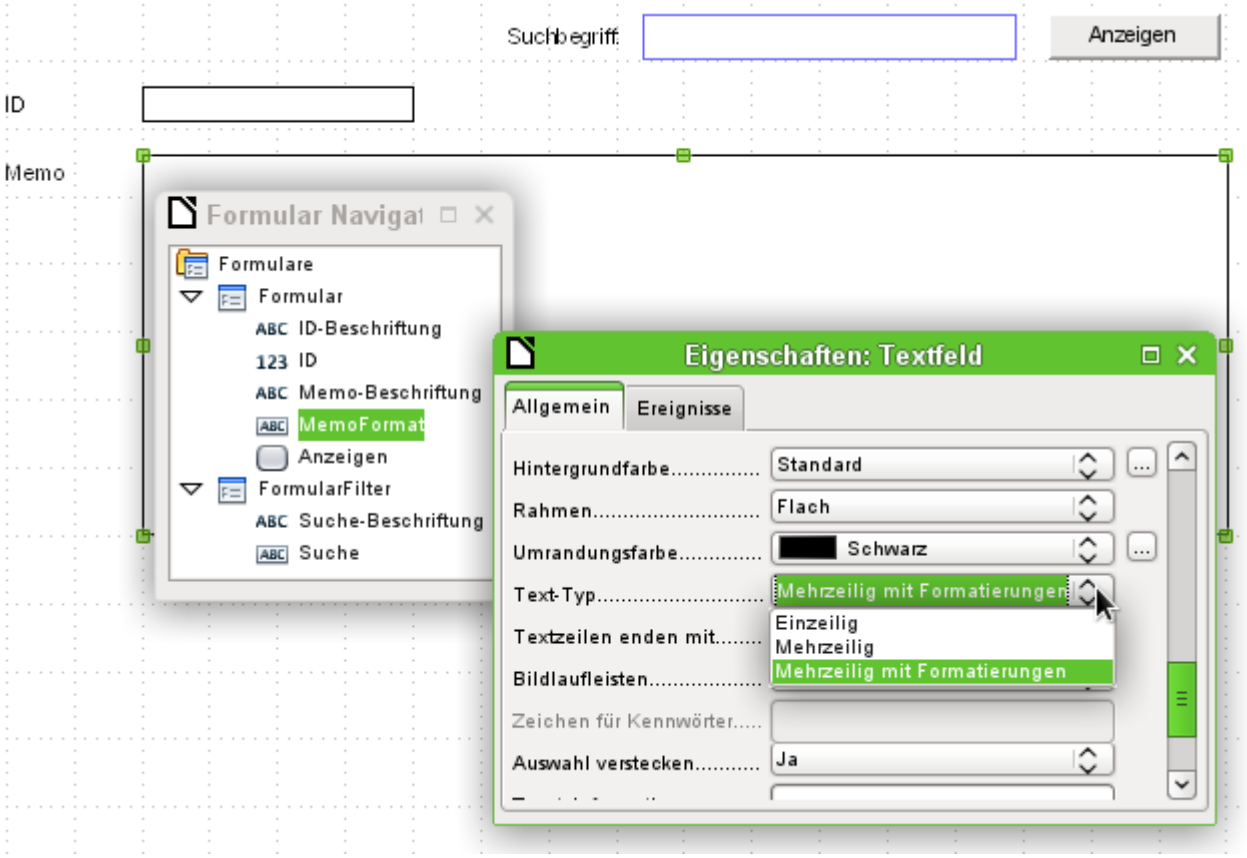
```
            oSelection.Min = len(oFeld.Text)
            oSelection.Max = len(oFeld.Text)
            oFeld.setFocus
            oFeld.Selection = oSelection
        END IF
    END IF
END SUB
```

Suche_markieren

Bei größeren Inhalten eines Textfeldes ist oft unklar, an welcher Stelle denn nun die Suche den Treffer zu verzeichnen hat. Da wäre es doch gut, wenn das Formular den entsprechenden Treffer auch markieren könnte. So sollte das dann im Formular aussehen:



Die Suche in diesem Formular baut auf eine der beiden Abfragen auf, wie sie weiter oben vorgestellt wurden. Um eine farbliche Markierung der Begriffe in dem Text zu erhalten bedarf es ein paar zusätzlicher Eingriffe in die Trickkiste.



In dem Formular sind neben dem Feld «ID» für den Primärschlüsseleintrag nur ein Feld «MemoFormat», das über **Eigenschaften → Allgemein → Text-Typ → Mehrzeilig mit Formatierungen** so eingestellt ist, dass es überhaupt Farben innerhalb von schwarzem Text darstellen kann. Die genaue Betrachtung der Eigenschaften des Textfeldes zeigt, dass der Reiter **Daten** fehlt. Daten lassen sich über ein Feld, das zusätzlich formatierbar ist, nicht eingeben. Das ist wohl dadurch begründet, dass die Datenbank selbst auch solche Formatierungen nicht speichert. Und trotzdem ist es durch den entsprechenden Makroeinsatz möglich, Text in dieses Feld hinein zu bekommen, ihn zu markieren und bei Änderungen auch wieder aus dem Feld hinaus in die Datenbank zu befördern.

Die Prozedur «InhaltUebertragen» dient dazu, den Inhalt aus dem Datenbankfeld "Memo" in das formatierbare Textfeld «MemoFormat» zu übertragen und so zu formatieren, dass bei einem entsprechenden Eintrag im Suchfeld der dazugehörige Begriff hervorgehoben wird.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Nach dem Datensatzwechsel**

```
Sub InhaltUebertragen(oEvent AS OBJECT)
    DIM inMemo AS INTEGER
    DIM oFeld AS OBJECT
    DIM stSuchtext AS STRING
    DIM oCursor AS OBJECT
    DIM inSuch AS INTEGER
    DIM inSuchAlt AS INTEGER
    DIM inLen AS INTEGER
    oForm = oEvent.Source
    inMemo = oForm.findColumn("Memo")
    oFeld = oForm.getByName("MemoFormat")
    oFeld.Text = oForm.getString(inMemo)
```

Zuerst werden die Variablen definiert. Anschließend wird über das Formular das Tabellenfeld "Memo" gesucht und aus diesem Feld über **`getString`**() der entsprechende Text des Feldes "Memo" der Tabelle "Tabelle" ausgelesen. Der entsprechende Feldinhalt wird in das Feld übertragen, das sich formatieren lässt, aber keine Verbindung zur Datenbank hat: «MemoFormat».

Bei Test ist es zuerst vorgekommen, dass sich das Formular zwar öffnete, aber leider die Formularleiste am unteren Rand des Formulars nicht mehr aufgebaut wurde. Deswegen erfolgt hier ein sehr kurzer Wartebefehl von 5/1000 Sekunden. Danach wird aus dem parallel zum «Formular» liegenden «FormularFilter» der angezeigte Inhalt als Suchtext ausgelesen.

```
Wait 5
stSuchtext = oForm.Parent.getByName("FormularFilter").getByName("Suche").Text
```

Um Textteile formatieren zu können muss ein (nicht sichtbarer) **`TextCursor`** in dem Feld erstellt werden, das den Text enthält. Die Darstellung des Textes in der Standardversion hat eine serifenbetonte Schriftart in 12-Punkt-Größe, die in anderen Formularteilen nicht unbedingt vorkommt und über das Formularfeld auch nicht direkt abwählbar ist. In dieser Prozedur wird direkt zu Beginn der Text einmal auf die gewünschte Darstellungsart eingestellt. Erfolgt dies nicht schon zu Beginn, so wird wegen der unterschiedlichen Formatierungen der oberer Textrand in dem Feld erst einmal angeschnitten. Die erste Zeile war in Versuchen nur zu 2/3 lesbar.

Damit der Cursor (wieder nicht sichtbar) den Text markiert wird er zuerst an den Anfang gesetzt und mit dem Zusatz **`true`** weiterbewegt zum Endpunkt, der ebenfalls den Zusatz **`true`** hat. Dann erfolgt die Zuweisung der notwendigen Eigenschaften wie Schriftgröße, Schriftstil, Schriftfarbe oder auch Schriftdicke. Anschließend wird der Cursor wieder zur Startposition gesetzt.

```
oCursor = oFeld.createTextCursor()
oCursor.gotoStart(true)
oCursor.gotoEnd(true)
oCursor.CharHeight = 10
oCursor.CharFontName = "Arial, Helvetica, Tahoma"
oCursor.CharColor = RGB(0,0,0)
oCursor.CharWeight = 100.000000   'com::sun::star::awt::FontWeight
oCursor.gotoStart(false)
```

Enthält das Feld Text und ist ein Eintrag zum Suchen vorhanden, so wird jetzt der Text nach dem Suchbegriff durchsucht. Die äußere Schleife fragt erst einmal nur nach der Bedingung, die nächste Schleife klärt noch einmal, ob der Suchtext denn tatsächlich in dem Text enthalten ist, der in «MemoFormat» steht. Diese Einstellung könnte auch unterlassen werden, da die Abfrage, auf der das Formular basiert, nur solchen Text anzeigt, auf den diese Bedingung zutrifft.

```
IF oFeld.Text <> "" AND stSuchtext <> "" THEN
   IF inStr(oFeld.Text, stSuchtext) THEN
      inSuch = 1
      inSuchAlt = 0
      inLen = Len(stSuchtext)
```

Der Text wird nach dem Suchtext durchsucht. Dies erfolgt in einer Schleife, die dann endet, wenn keine weitere Trefferposition mehr angezeigt wird. **`InStr`**() liefert dabei die Fundstelle des ersten Zeichens des Suchtextes, in der aufgezeigten Fassung unabhängig von Groß- und Kleinschreibung. Die Schleife wird dadurch gesteuert, dass der Suchbeginn **`inSuch`** bei jedem Schleifenende in der Summe um 1 erhöht wird (erste Schleifenzeile -1, letzte Schleifenzeile +2). Bei jedem Durchgang wird der Cursor mit **`oCursor.goRight(Position, false)`** zuerst ohne zu markieren an die Startstelle gesetzt, dann um die Länge des Suchtextes weiter mit der Markierungsaufforderung nach rechts gesetzt. Dann wird die gewünschte Formatierung (blau, etwas dicker) vorgenommen und der Cursor wieder für den nächsten Start an den Startpunkt der Markierung zurückgesetzt.

```
      DO WHILE inStr(inSuch, oFeld.Text, stSuchtext) > 0
         inSuch = inStr(inSuch, oFeld.Text, stSuchtext) - 1
         oCursor.goRight(inSuch-inSuchAlt,false)
         oCursor.goRight(inLen,true)
         oCursor.CharColor = RGB(102,102,255)
         oCursor.CharWeight = 110.000000
         oCursor.goLeft(inLen,false)
         inSuchAlt = inSuch
         inSuch = inSuch + 2
      LOOP
   END IF
END IF
```

```
End Sub
```

Die Prozedur «InhaltSchreiben» dient dazu, den Inhalt aus dem formatierbaren Textfeld «Memo-Format» in die Datenbank zu übertragen. Dies erfolgt in dieser Fassung unabhängig davon, ob eine Änderung vorgenommen würde.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Vor dem Datensatzwechsel**

```
Sub InhaltSchreiben(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM inMemo AS INTEGER
    DIM loID AS LONG
    DIM oFeld AS OBJECT
    DIM stMemo AS STRING
    oForm = oEvent.Source
    IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
```

Das auslösende Ereignis ist doppelt belegt. Nur der Implementationsname, der auf **`ODatabaseForm`** endet, gibt des richtigen Zugriff auf den Datensatz.

```
        IF NOT oForm.isBeforeFirst() AND NOT oForm.isAfterLast() THEN
```

Beim Einlesen, auch beim Reload des Formulars, steht der Cursor vor dem ersten Datensatz. Würde jetzt ein Schreibversuch unternommen, dann erscheint die Meldung «ungüliger Cursorstatus».

```
            inMemo = oForm.findColumn("Memo")
            loID = oForm.findColumn("ID")
            oFeld = oForm.getByName("MemoFormat")
            stMemo = oFeld.Text
            IF stMemo <> "" THEN
                oForm.updateString(inMemo,stMemo)
            END IF
            IF stMemo <> "" AND oForm.getString(loID) <> "" THEN
                oForm.UpdateRow()
            END IF
        END IF
    END IF
End Sub
```

Das Tabellenfeld "Memo" wird aus der Datenquelle des Formulars herausgesucht. Ebenso das Feld "ID". Befindet sich im Feld «MemoFormat» Text, so wird er mit **`oForm.updateString`**() in das Feld "Memo" der Datenquelle übertragen. Nur wenn bereits ein Eintrag im Feld "ID" existiert, also der Primärschlüssel belegt ist, erfolgt ein Update. Ansonsten wird ja sowieso ein neuer Datensatz über die Formularfunktionen eingefügt, da das Formular die Änderung entsprechend bemerkt und eine Abspeicherung selbständig vornimmt.

Rechtschreibkontrolle_hinterher

Um dieses und die folgenden Formulare zusammen mit einer Datenbank lauffähig zu halten muss wie bei der Suchmarkierung für das Memo-Feld **Eigenschaften → Allgemein → Text-Typ → Mehrzeilig mit Formatierungen** gewählt werden. Ohne die Formatierungen ist es nicht möglich, in dem Textfeld eine geschlängelte Linie zu erreichen.

Um den Kontakt zwischen der Tabelle und dem nicht verknüpften Formularfeld herzustellen, muss beim Formular über **Ereignisse → Vor dem Datensatzwechsel** das Makro «InhaltSchreiben» ausgelöst werden. Das Makro wurde im Kapitel *Suche_markieren* vorgestellt. An **Ereignisse → Nach dem Datensatzwechsel** wird das Makro «InhaltUebertragenOhneSuche» aufgerufen. Dies entspricht dem Makro «InhaltUebertragen». Lediglich die Schleife zur Markierung der Suchbegriffe wird hier ausgespart.

Das Makro, mit dem die Markierung fehlerhafter Worte erfolgt, ist an das **Textfeld → Ereignisse → Modifiziert** gebunden. Die Veränderung wird hier erst wahrgenommen, wenn sich der Cursor außerhalb des Formularfeldes befindet, also z.B. ein Klick mit der Maus auf den Hintergrund

erfolgt. Dies liegt daran, dass keine Verbindung des Kontrollfeldes zu einem Datenfeld des Formulars gegeben ist.

Das Makro schreibt den gesamten Text neu und unterstreicht mit einer roten Wellenlinie eventuelle Fehler. Es berücksichtigt dabei in dieser Fassung nicht Satzzeichen, die an ein Wort anschließen. Deshalb werden alle Worte mit anschließendem Satzzeichen ebenfalls als Fehler angesehen. Satzzeichen werden bei dem Makro zur direkten Rechtschreibkontrolle berücksichtigt.

```
SUB MarkierenFehlerhafterWorteWellenlinie(oEvent AS OBJECT)
    DIM aProp() AS NEW com.sun.star.beans.PropertyValue
    DIM n AS INTEGER
    DIM oFeld AS OBJECT
    DIM oCursor AS OBJECT
    DIM aText()
    DIM oLinguSvcMgr AS OBJECT
    DIM oSpellChk AS OBJECT
    DIM i AS INTEGER
    oFeld = oEvent.Source
    oCursor = oFeld.Model.createTextCursor()
```

Das Textfeld wird direkt durch das auslösende Ereignis ermittelt. In dem Textfeld wird ein Textcursor erstellt. Enthält das Feld Text, dann wird der gesamte Text in ein Array aufgesplittet. Die Trennung erfolgt nach den Leerzeichen.

```
    IF oFeld.text <> "" THEN
        aText = Split(oFeld.text)
        oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
        IF NOT IsNull(oLinguSvcMgr) THEN
            oSpellChk = oLinguSvcMgr.getSpellChecker()
        END IF
```

Die Rechtschreib-Prüfung erfolgt mit Hilfe des **`LinguServiceManager`**. Jedes einzelne Wort wird in der folgende Schleife daraufhin überprüft, ob es in dem entsprechenden Wörterbuch verzeichnet ist. Ist das Wort in dem deutschsprachigen Wörterbuch **`de`** verzeichnet, so wird auf die Funktion **`isValid`** mit einem **`True`** geantwortet. Ansonsten ist das Wort nicht gültig. Es erfolgt dann die Kennzeichnung mit einer Wellenlinie **`9`** und der entsprechenden roten Farbgebung.

```
        FOR i = LBound(aText) TO UBound(aText)
            oCursor.gotoEnd(true)
            IF NOT oSpellChk.isValid(aText(i), "de", aProp()) THEN
                oCursor.CharUnderline = 9
                oCursor.CharUnderlineHasColor = True
                oCursor.CharUnderlineColor = RGB(255,51,51)
            END IF
            oFeld.Model.insertString(oCursor,aText(i),true)
```

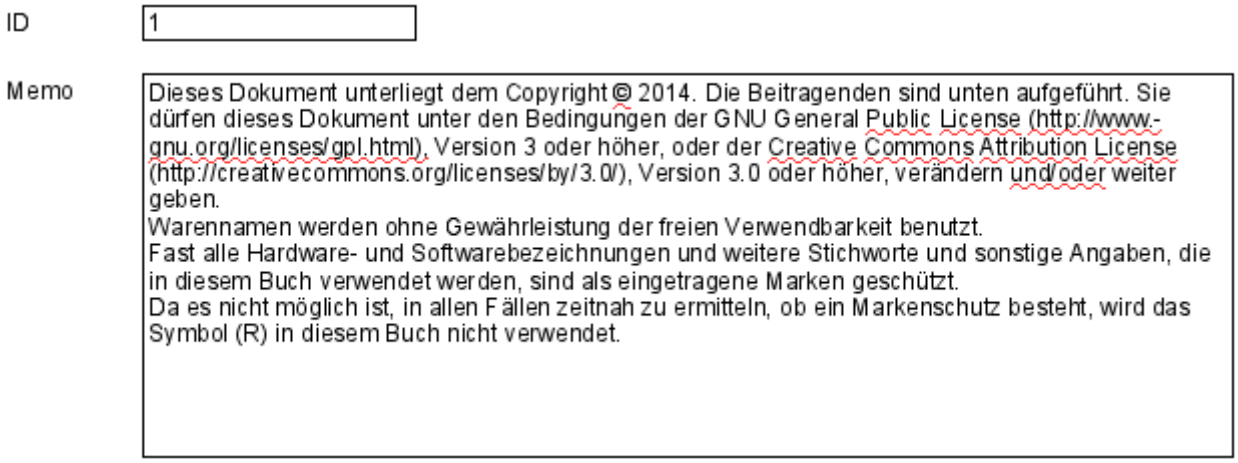
Anschließend wird der entsprechende Text mit dieser Formatierung oder eben ohne Formatierung in das Textfeld eingefügt. Solange noch weiter Text in dem Array vorhanden ist wird anschließend ein Leerzeichen angefügt, bei dem auf jeden Fall die Unterstreichung mit **`CharUnderline = 0`** aufgehoben wird. Sonst würde nach dem ersten als falsch erkannten Wort der gesamte Text mit roten Wellenlinien unterstrichen.

```
            IF i < UBound(aText) THEN
                oCursor.gotoEnd(true)
                oCursor.CharUnderline = 0
                oFeld.Model.insertString(oCursor," ",true)
            END IF
        NEXT
    END IF
END SUB
```

Rechtschreibkontrolle_direkt

Auf **mehrzeilige Textfelder mit Formatierungen** greift auch dieses Makro zu. Entsprechend muss auch, wie bei dem vorherigen Kapitel, der Inhalt bei jedem Datensatzwechsel zuerst geschrieben und danach der Inhalt des neuen Datensatzes in das Formularfeld geladen werden.

Die Prozeduren «InhaltUebertragenOhneSuche» und «InhaltSchreiben» werden also auch hier benötigt.



Die Rechtschreibkontrolle wird in dem obigen Formular dadurch ausgelöst, dass in dem Formularfeld entweder eine Leertaste oder ein Return betätigt wird. Sie läuft also nach der Beendigung eines Wortes jedes Mal ab und könnte gegebenenfalls auch noch mit dem Fokusverlust des Formularfeldes gekoppelt werden, damit auch das letzte Wort sicher überprüft wird.

Die Prozedur zur Rechtschreibkontrolle ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Taste losgelassen**

```
SUB MarkierungFehlerDirekt(oEvent AS OBJECT)
    GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

Es wird die Funktion **`RTrimStr`** zum Entfernen von Satzzeichen am Ende vor Worten benötigt. Sonst werden alle Worte, denen ein Komma, Punkt oder irgendein anderes Satzzeichen folgt, als falsch angesehen.

```
    DIM aProp() AS NEW com.sun.star.beans.PropertyValue
    DIM oLinuSvcMgr AS OBJECT
    DIM oSpellChk AS OBJECT
    DIM oFeld AS OBJECT
    DIM arText()
    DIM stWort AS STRING
    DIM inlenWort AS INTEGER
    DIM ink AS INTEGER
    DIM i AS INTEGER
    DIM oCursor AS OBJECT
    oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
    IF NOT IsNull(oLinguSvcMgr) THEN
        oSpellChk = oLinguSvcMgr.getSpellChecker()
    END IF
```

Zuerst werden alle Variablen deklariert. Danach wird auf das Rechtschreibüberprüfungsmodul **`SpellChecker`** zugegriffen. Mit diesem Modul werden anschließend die einzelnen Worte auf ihre Richtigkeit hin überprüft.

```
    oFeld = oEvent.Source.Model
    ink = 0
    IF oEvent.KeyCode = 1280 OR oEvent.KeyCode = 1284 THEN
```

Das Ereignis, das das Makro auslöst, ist ein Tastendruck. Zu dem Ereignis wird ein Code für jede Taste mitgeliefert, der **`KeyCode`**. Der **`KeyCode`** für die Returntaste ist 1280, der für die Leertaste ist 1284. Wie viele andere Informationen sind diese Informationen einfach durch das Tool «Xray» gewonnen worden. Wird also eine Leertaste oder die Returntaste betätigt, so wird die Rechtschreibung überprüft. Sie startet also zu jedem Wortende. Lediglich die Überprüfung für das letzte Wort ist so nicht automatisch möglich.

Bei jedem Durchlauf werden alle Worte des Textes überprüft. Die Überprüfung einzelner Worte könnte eventuell auch möglich sein, bedeutet aber erheblich mehr Aufwand.

Der Text wird also in Worte aufgesplittet. Trenner ist hier das Leerzeichen.

```
        arText = Split(RTrim(oFeld.Text)," ")
        FOR i = LBound(arText) TO UBound(arText)
            stWort = arText(i)
            inlenWort = len(stWort)
            stWort = Trim( RtrimStr( RtrimStr( RtrimStr( RtrimStr(
                RtrimStr(stWort,","), "."),"?"),"!"),"."))
```

Das einzelne Wort wird ausgelesen, seine ungekürzte Länge ist notwendig für die folgenden Bearbeitungsschritte. Nur so kann die Position des Wortes innerhalb des gesamten Textes bestimmt werden, die auch für die gezielte Markierung von Schreibfehlern gebraucht wird.

Mit **`Trim`** werden Leerzeichen entfernt, mit der Funktion **`RTrimStr`** Kommas und Satzzeichen am Ende des Textes.

```
            IF stWort <> "" THEN
                oCursor = oFeld.createTextCursor()
                oCursor.gotoStart(false)
                oCursor.goRight(ink,false)
                oCursor.goRight(inLenWort,true)
                If Not oSpellChk.isValid(stWort, "de", aProp()) Then
                    oCursor.CharUnderline = 9
                    oCursor.CharUnderlineHasColor = True
                    oCursor.CharUnderlineColor = RGB(255,51,51)
                ELSE
                    oCursor.CharUnderline = 0
                END IF
            END IF
            ink = ink + inLenWort + 1
        NEXT
    END IF
END SUB
```

Hat das Wort einen Inhalt, so wird zuerst einmal ein Textcursor erstellt. Der Textcursor wird ohne Markierung an den Start des Textes in dem Eingabefeld gesetzt. Dann geht es, immer noch ohne Markierung, um den Betrag nach rechts im Text vorwärts, der in der Variablen **`ink`** gespeichert ist. Diese Variable ist am Anfang 0, nach Durchlaufen der ersten Schleife dann so groß wie das vorhergehende Wort lang war +1 für das angehängte Leerzeichen. Dann wird der Cursor mit Markierung um die Länge des aktuellen Wortes weiter gesetzt. Erfolgt jetzt eine Änderung der Buchstabeneigenschaften, so betrifft diese nur den markierten Bereich.

Der **`Spellchecker`** startet. Als Variablen müssen das Wort und der Landescode übergeben werden. Ohne Landescode ist alles richtig. Das Array bleibt in der Regel leer.

Ist das Wort nicht in den Lexika eingetragen, so wird es mit einer roten Wellenlinie versehen. Die Wellenlinie entspricht hier der **`'9'`**. Ist das Wort eingetragen, so wird statt einer Wellenlinie keine Linie (**`'0'`**) gezeichnet. Dieser Schritt ist notwendig, weil sonst ein einmal als falsch erkanntes Wort bei einer Korrektur auch weiterhin mit der roten Wellenlinie gekennzeichnet würde. Eine rote Wellenlinie würde nie aufgehoben, da es keine entgegengesetzte Formatierung gibt.

Terminübersicht am Beispiel einer Ferienhausvermietung

Die einfache Eingabe von Daten für die Belegung eines Ferienhauses ist in Base problemlos möglich. Wie aber erhalte ich einen Überblick, zu welchen Terminen das Ferienhaus noch frei ist? Dazu muss eine Jahresübersicht erstellt werden, die am besten alle Datumswerte des Jahres enthält. Wie gebe ich jetzt sicher ein, dass ein Ferienhaus nicht doppelt belegt wird. Auch hierzu bietet das Formular dieser Datenbank einen Lösungsvorschlag.

Tabellen

Die Tabelle "Mieter" enthält in diesem Fall lediglich die Namen der Mieter. Hier muss natürlich eine komplette Adressverwaltung eingebaut werden, damit die Datenbank einen Sinn macht.

In der Tabelle "Mieter" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann als Auto-Wert-Feld gesetzt werden.
Name	Text	Name des Mieters. **Feldeigenschaften → Eingabe erforderlich → Ja**

Die Tabelle "Haus" enthält in diesem Fall lediglich die Bezeichnung des Hauses. Hier sollten entsprechend noch Details wie die Adresse, die Anzahl der möglichen Personen usw. mit aufgenommen werden, wenn eine Datenbank mit dem entsprechenden Vermietungsziel erstellt wird.

In der Tabelle "Haus" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld kann als Auto-Wert-Feld gesetzt werden.
Haus	Text	Bezeichnung des Hauses. **Feldeigenschaften → Eingabe erforderlich → Ja**

Die Tabelle "Reservierung" verbindet die Tabelle "Mieter" mit der Tabelle "Haus".

In der Tabelle "Reservierung" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
Haus_ID	Integer	Fremdschlüssel der Tabelle "Haus". Gleichzeitig einer der drei Primärschlüssel dieser Tabelle.
Mieter_ID	Integer	Fremdschlüssel der Tabelle "Mieter". Gleichzeitig einer der drei Primärschlüssel dieser Tabelle.
Startdatum	Datum	Nimmt den Beginn der Mietperiode auf. Muss zusammen mit den vorhergehenden Feldern ein Primärschlüssel sein, da sonst ein Mieter nicht zweimal das gleiche Haus mieten kann.
Enddatum	Datum	Hier wird das Ende der Mietperiode verzeichnet. **Feldeigenschaften → Eingabe erforderlich → Ja**

Das Ende der Mietperiode könnte noch mit einer Bedingung so versehen werden, dass das eingegebene Datum auf jeden Fall nach dem Startdatum liegt. Hier wurde darauf verzichtet, da die Eingabe komplett über das Formular gesteuert wird.

Die drei oben genannten Tabellen sind die Tabellen, in denen beständig Eingaben erfolgen sollen. Daneben existiert noch wieder eine Tabelle "Filter", die dazu dient, nach bestimmten Zeiträumen die Eingaben im Formular zu filtern.

Aus der Tabelle "NrBis31" wurde der Primärschlüssel nach der Eingabe der Ziffern entfernt, damit nicht irrtümlich eine der Zahlen gelöscht wird. Solch eine Tabelle kann auch einfach über Calc und die automatische Nummerierung erzeugt werden. Die Tabelle enthält nur eine Spalte, nämlich die Spalte "Nr". Mit Hilfe dieser Tabelle wird eine fortlaufende Datumsübersicht erzeugt.

Abfragen

Die Abfragen greifen auf zwei unterschiedliche Ansichten zu. Die Ansichten dienen dazu, laufende Datumswerte wieder zu geben bzw. auch noch mit einer laufenden Nummerierung zu versehen.

Ansicht_Datum

```
SELECT DISTINCT
   CAST( "Y"."Nr" + (SELECT YEAR(IFNULL("BeginnDatum",CURRENT_DATE))
         FROM "Filter" WHERE "ID" = True) - 2 || '-' ||
      CASEWHEN
      ( "M"."Nr" < 10, '0' || "M"."Nr", '' || "M"."Nr" ) || '-' ||
      CASEWHEN
      ( "D"."Nr" < 10, '0' || "D"."Nr", '' || "D"."Nr" )
      AS DATE )
   AS "Datum"
FROM "NrBis31" AS "D", "NrBis31" AS "M", "NrBis31" AS "Y"
WHERE
   "Y"."Nr" <= (SELECT YEAR(IFNULL("BeginnDatum",CURRENT_DATE)) -
      YEAR(IFNULL("EndeDatum",CURRENT_DATE)) + 3
      FROM "Filter" WHERE "ID" = True)
   AND "M"."Nr" <= 12
   AND "D"."Nr" <= 31
```

Die Tabelle "NrBis31" enthält lediglich die Zahlen von 1 bis 31. Aus diesen Zahlen wird ein Datumswert zusammengesetzt. Das Datum muss dabei in der Reihenfolge YYYY-MM-DD vorliegen. Erst dann kann es durch die HSQLDB in einen Datumswert umgewandelt werden.

Die Tabelle "NrBis31" wird dreimal durch einen Alias eingebunden. Da zwischen den verschiedenen Alias-Bezeichnungen keine Beziehung aufgezeigt wird ermittelt die Abfrage alle Kombinationen der Werte, die aus der Tabelle zu entnehmen sind. Das Jahr beginnt mit Werten, die ein Jahr vor dem Jahr liegen, in dem das "BeginnDatum" der Tabelle "Filter" liegt. Es sollen alle Jahre bis 1 Jahr nach dem "EndeDatum" aus der Tabelle "Filter" wieder gegeben werden. Die Spannweite wird durch die Jahresdifferenz zwischen den beiden Datumswerten ermittelt und entsprechend um 3 erhöht, da gleiche Jahreszahlen ja sonst gar keine Jahreswiedergabe ermöglichen würden. Für den Monat werden die Werte von 1 bis 12 ausgelesen und für den Tag die Werte von 1 bis 31.

Das Startjahr wird aus dem "BeginnDatum" der Tabelle "Filter" ausgelesen. Das Startjahr wird zu dem jeweiligen Datensatz aus "NrBis31" addiert. Da die Zahlenwerte aus der Tabelle "NrBis31" mit 1 beginnen und die Jahreszahlen 1 Jahr vor dem Jahr beginnen sollen, das aktuell angezeigt wird, wird von dem Betrag des Startjahres 2 subtrahiert.

Ist die Monatszahl kleiner als 10, so wird eine '0' vor die Monatszahl gesetzt. Ansonsten wird die Monatszahl direkt nach dem '-' an die Jahreszahl angefügt. Entsprechend wird auch mit der Tageszahl umgegangen.

Insgesamt ergibt sich daraus ein Datum in Textform, der jetzt noch in einen Datumswert über **CAST … AS DATE** umgewandelt wird.

Durch die Umwandlung erstellt die HSQLDB aus einem Datumswert wie z.B. 31.02.2015 stattdessen den 3.03.2015. Schließlich hat der Februar nicht 31 Tage. Tageswerte über 31 akzeptiert die HSQLDB allerdings nicht. Sonst wäre über diesen Weg auch leicht eine Addition von Tagen wie mit **DATEADD** in anderen Datenbanken zu einem Datum möglich. Diese Umwandlung erzeugt jetzt bei

einigen Datumswerten doppelte Einträge wie eben bei dem 3.03.2015. Diese Doppler sollen unterbunden werden. Das geschieht schließlich mit dem Zusatz **DISTINCT**.

Die Ansicht wird von der HSQLDB direkt in der korrekten Datumsreihenfolge ausgegeben.

Ansicht_Datum_lfdNr

```
SELECT "a"."Datum",
   (SELECT COUNT(*) FROM "Ansicht_Datum" WHERE "Datum" <= "a"."Datum")
   AS "lfdNr"
FROM "Ansicht_Datum" AS "a"
```

Diese Ansicht greift auf die vorher erstellte Ansicht zu. Sie fügt allerdings noch eine Spalte hinzu, die eine fortlaufende Nummerierung der der Datumswerte erstellt. Es werden durch eine korrelierende Unterabfrage alle Datumswerte gezählt, die kleiner oder gleich dem aktuellen Datumswert des angezeigten Datensatzes sind.

Eine solche Auflistung mit fortlaufender Nummerierung macht es möglich, zu einem Datum eine bestimmte Anzahl von Tagen zu addieren oder zu subtrahieren und als Ergebnis wieder ein Datum zu erhalten. Hierdurch kann also in Grenzen die Funktion **DATEADD** nachgestellt werden.

Belegung

Aus den Eingaben in die Reservierungstabelle soll ein Überblick geschaffen werden, zu welchen Zeitpunkten bestimmte Häuser noch frei sind. Der Screenshot zeigt ein entsprechendes Abfrageergebnis.

Datum	Haus1	Haus2	Haus3	Haus4	Haus5	Haus6	Haus7	Haus8
01.01.15	belegt		belegt					
02.01.15	belegt		belegt					
03.01.15	belegt		belegt					
04.01.15	belegt		belegt					
05.01.15	belegt		belegt					
06.01.15	belegt	belegt						
07.01.15	belegt	belegt						
08.01.15		belegt						
09.01.15	belegt	belegt						
10.01.15	belegt	belegt	belegt					
11.01.15	belegt	belegt	belegt					
12.01.15	belegt	belegt	belegt					
13.01.15	belegt	belegt	belegt					
14.01.15	belegt	belegt	belegt					
15.01.15	belegt	belegt	belegt					
16.01.15	belegt	belegt	belegt					
17.01.15	belegt	belegt						
18.01.15	belegt	belegt						
19.01.15	belegt	belegt						
20.01.15	belegt	belegt						
21.01.15	belegt							
22.01.15	belegt							

In der ersten Spalte wird das Datum aus der "Ansicht_Datum" ausgelesen. Dabei ist durch Einträge in der Tabelle "Filter" festgelegt, von welchem bis zu welchem Datum die Anzeige erfolgen soll. Die gesamte Abfrage bezieht sich nur auf "Ansicht_Datum". In einer korrelierenden Unterabfrage wird dann für jedes Haus abgefragt, ob das Datum der ersten Abfragespalte Zwischen "Startdatum" und Enddatum" eines Eintrags zu finden ist, der in der Tabelle "Reservierung" existiert und dem jeweiligen Primärschlüssel des Hauses entspricht.

```
SELECT "a"."Datum",
   ( SELECT 'belegt' FROM "Reservierung" WHERE "Haus_ID" = 1
      AND "a"."Datum" BETWEEN "Startdatum" AND "Enddatum" )
      AS "Haus1",
   ( SELECT 'belegt' FROM "Reservierung" WHERE "Haus_ID" = 2
```

```
      AND "a"."Datum" BETWEEN "Startdatum" AND "Enddatum" )
      AS "Haus2"
   ...
      AS "Haus8"
FROM "Ansicht_Datum" AS "a"
WHERE "a"."Datum"
   BETWEEN IFNULL(
      ( SELECT "BeginnDatum" FROM "Filter" WHERE "ID" = TRUE ),
      CURRENT_DATE )
   AND IFNULL(
      ( SELECT "EndeDatum" FROM "Filter" WHERE "ID" = TRUE ),
      CURRENT_DATE )
```

Die Abfrage hat noch den Nachteil, dass der Name des Hauses direkt in der Abfrage enthalten ist. Hier müsste also die Hausbezeichnung auch im Abfragecode angepasst werden, wenn sie nicht der Namensgebung in der Tabelle "Haus" entspricht. Innerhalb eines Formulars ist die Benennung natürlich über das entsprechende Tabellenkontrollfeld zu ändern. Es ließe sich dort auch über eine einzeilige Abfrage der Name des Hauses direkt auslesen.

Bericht

Diese Abfrage dient als Basis für den Bericht. Hier sollen Hausbezeichnung und Name des Mieters neben den Informationen aus der Tabelle "Reservierung" erscheinen. Außerdem kommen noch Informationen hinzu, die die Mietdauer und den Termin betreffen, zu dem eine Anzahlung zu erfolgen hat. Die letzte Spalte in dem folgenden Screenshot ist in diesem Fall vorformatiert worden, so dass tatsächlich ein Datum sichtbar wird. Direkt ausgeführt erscheint dort nur eine Zahlenkette, die die Tage seit dem 30.12.1899 (0-Position) zählt, sichtbar.

Haus	Name	Startdatum	Enddatum	Tage	AnzahlungBis
Haus 1	Dompapst	01.01.15	07.01.15	6	02.12.14
Haus 1	Schulze	09.01.15	16.01.15	7	10.12.14
Haus 1	Appelkorn	17.01.15	31.01.15	14	18.12.14
Haus 2	Müller	06.01.15	20.01.15	14	07.12.14
Haus 3	Maier	01.01.15	05.01.15	4	02.12.14
Haus 3	von Weich	10.01.15	16.01.15	6	11.12.14

```
SELECT "Haus"."Haus", "Mieter"."Name",
   "a"."Startdatum", "a"."Enddatum",
   DATEDIFF( 'dd', "a"."Startdatum", "a"."Enddatum" ) AS "Tage",
   ( SELECT "Datum" FROM "Ansicht_Datum_lfdNr" WHERE "lfdNr" =
      ( SELECT "lfdNr" FROM "Ansicht_Datum_lfdNr" WHERE "Datum" =
      "a"."Startdatum" ) - 30 )
      AS "AnzahlungBis"
FROM "Reservierung" AS "a", "Haus", "Mieter"
WHERE "a"."Haus_ID" = "Haus"."ID"
   AND "a"."Mieter_ID" = "Mieter"."ID"
ORDER BY "Haus"."Haus" ASC, "a"."Startdatum" ASC
```

Da auf die Tabelle "Reservierung" wegen einer korrelierenden Unterabfrage durch ein Alias zugegriffen werden muss, werden alle damit verbundenen Felder über das Alias "a" angesprochen. Die Verbindung der Tabellen entspricht der Verbindung innerhalb der erklärten Beziehungen dieser Datenbank. Die ersten vier Felder dieser Abfrage sind also leicht über die GUI zusammen zu klicken. Die Tage, die ein Mieter in einem Haus ist, werden über die Funktion **`DATEDIFF`** berechnet. **`'dd'`** berechnet hier den Unterschied in Tagen zwischen den folgenden beiden Datumswerten.

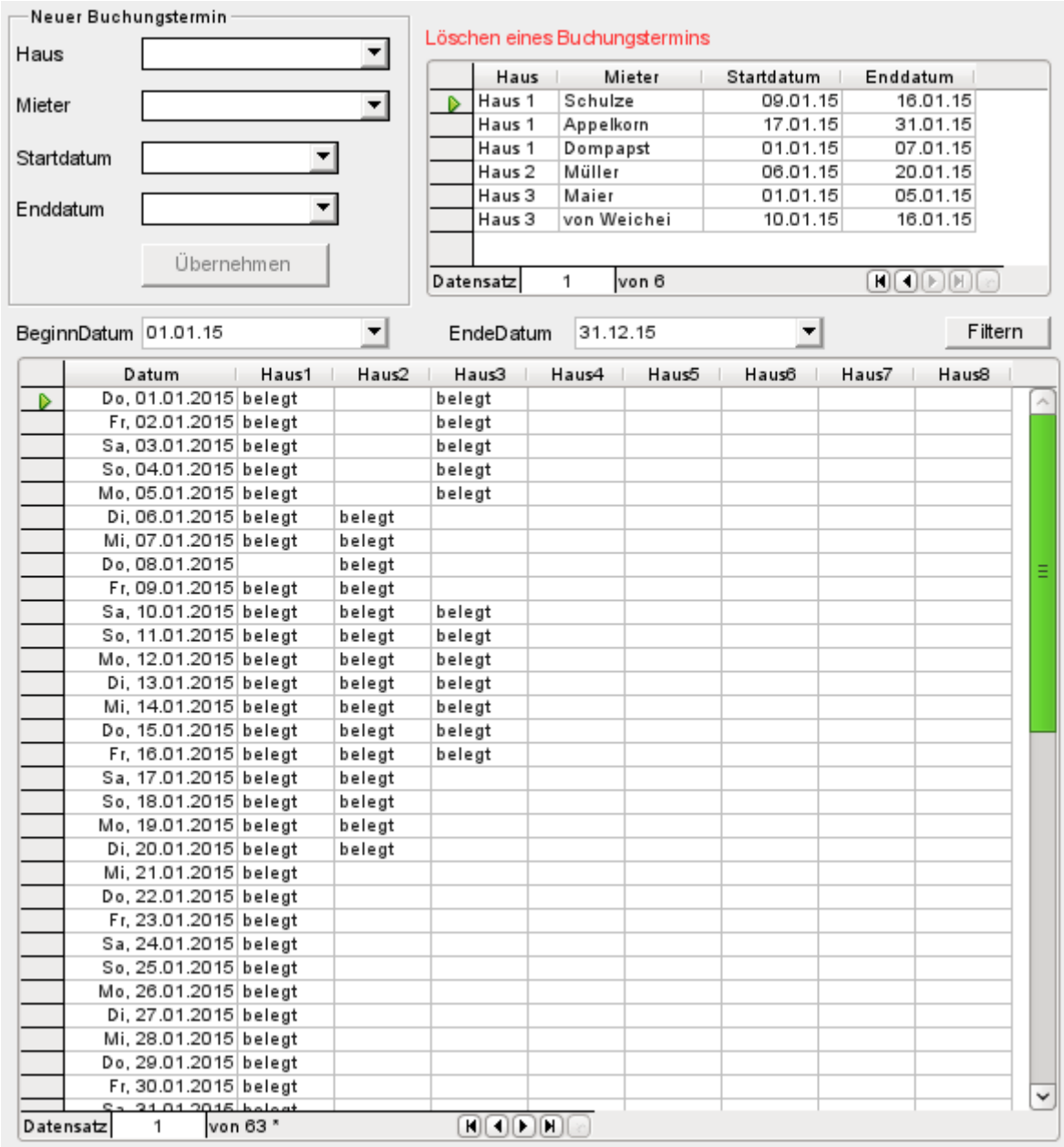
Um das Datum zu berechnen, bis zu dem die Anzahlung fällig ist, wird auf die "Ansicht_Datum_lfdNr" zugegriffen. Durch diese Unterabfrage wird die fehlende Funktion **`DATEADD`** in diesem Falle ersetzt. Zu dem "Startdatum" des aktuellen Datensatzes wird die entsprechende "lfdNr" aus "Ansicht_Datum_lfdNr" ermittelt. Von dieser Nummer wird 30 als Anzahl der Tage sub-

trahiert, die das Anzahlungsdatum vor dem Startdatum liegen soll. Dann wird in der äußeren Abfrage zu dieser neuen "lfdNr" das entsprechende Datum aus der "Ansicht_Datum_lfdnr" ermittelt.

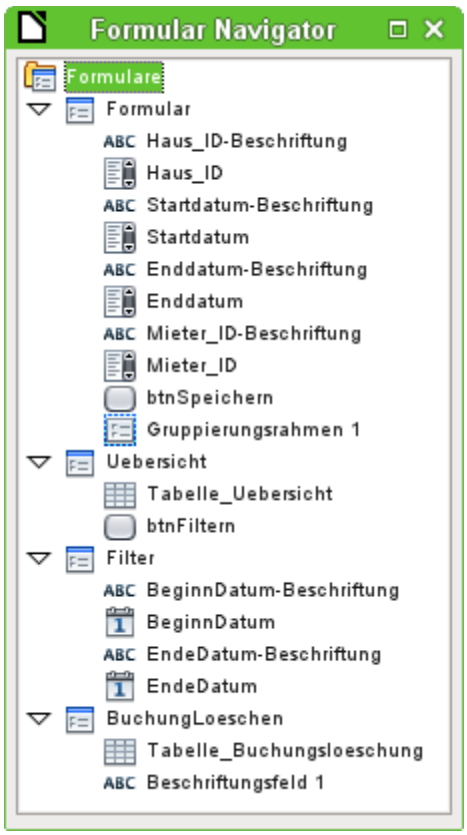
Diese Abfragetechnik funktioniert leider nur solange, wie beide Datumswerte in der Ansicht vertreten sind.

Formular

Das Formular gliedert sich in einen Eingabebereich für neue Buchungstermine, eine Auswahl der bestehenden Buchungstermine und eine Anzeige zur Übersicht über Buchungstermine in einem bestimmten Zeitraum.



Sämtliche Bereiche dieses Formulars sind in eigenständigen Formularen auf einer gemeinsamen Benutzeroberfläche zusammengefasst. Der Formularnavigator zeigt hier die entsprechenden Formularfelder auf:



Der größte Aufwand wird betrieben, um bei der Auswahl des Startdatums bzw. des Enddatums nur tatsächlich mögliche Werte bereit zu stellen. Aus diesem Grunde sind die Felder auch nicht als Datumsfelder, sondern als Listenfelder vorgegeben. Für die Listenfelder kann über den SQL-Code die Auswahlmöglichkeit des Inhaltes gesteuert werden. Beide Listenfelder sind darauf eingestellt, dass sie im SQL-Code den Wert des Feldes, den sie anzeigen, anschließend auch in die darunterliegende Tabelle übertragen: **Eigenschaften → Daten → gebundenes Feld → 0**.

Würde in obigem Beispiel über das Listenfeld für das Haus 'Haus 4' gewählt, dann sind für beide Datumsfelder sämtliche Eingaben offen. Das Enddatums-Feld darf allerdings nur eine Eingabe möglich machen, die mindestens einen Tag hinter dem Startdatum liegt.

Schwieriger wird es z.B. bei der Auswahl von 'Haus 3'. Dort sind im Listenfeld für das Startdatum nur die Werte möglich, die eben bisher nicht mit 'belegt' gekennzeichnet sind. Wird aber jetzt z.B. der 6.1.15 für dieses Haus gewählt, dann darf das Listenfeld für das Enddatum nur noch die Datumswerte vom 7.1. bis zum 9.1. anbieten. Es muss also zuerst das Haus, dann das Startdatum und erst danach das Enddatum gewählt werden. Bei der Auswahl des Hauses muss über **Eigenschaften → Ereignisse → Modifiziert** das Makro «Startdatum» für das Startdatum ausgelöst werden. Bei der Auswahl des Startdatums muss über **Eigenschaften → Ereignisse → Modifiziert** das Makro «Enddatum» für das Enddatum ausgelöst werden

```
SUB Startdatum(oEvent AS OBJECT)
    DIM oFeldStart AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM inID AS INTEGER
    DIM stSql(0) AS STRING
    oFeldStart = oEvent.Source.Model
    oForm = oFeldStart.Parent
    oFeld = oForm.getByName("Startdatum")
    inID = oFeldStart.CurrentValue
    stSql(0) = "SELECT ""Datum"" FROM ( SELECT ""a"".""Datum"",
       ( SELECT 'belegt' FROM ""Reservierung"" WHERE ""Haus_ID"" = '"+inID+"'"
       AND ""a"".""Datum"" BETWEEN ""Startdatum"" AND ""Enddatum"" ) AS ""Haus""
       FROM ""Ansicht_Datum"" AS ""a""
```

```
        WHERE ""a"".""Datum"" >= CURRENT_DATE) WHERE ""Haus"" IS NULL"
    oFeld.ListSource = stSql
    oFeld.refresh
END SUB
```

Über das auslösende Ereignis des Formularfeldes wird auf das Formular geschlossen. Auf diesem Wege wird dann auch die Verbindung zu dem Feld mit dem Namen «Startdatum» aufgebaut, das in dem Listenfeld den ermittelten SQL-Code in eine entsprechende Liste umwandeln soll.

Aus dem auslösenden Feld wird der Schlüsselwert für das Haus ermittelt. Der Wert wird in die Variable **`inID`** gelesen. Hier sollte bei größeren Zahlenwerten darauf geachtet werden, dass die Variable des Formats **`INTEGER`** in einem Makro (Basic) nicht dem Format INTEGER der HSQLDB entspricht. Statt 2^{32} kann INTEGER in Basic «nur» 2^{16} verschiedene Zahlenwerte darstellen. Für die Häuserzahl ist dies allerdings wohl nie von Belang.

Der SQL-Code für das Listenfeld muss als Wert eines Arrays eingelesen werden, da das Listenfeld als **`ListSource`** ein Array erwartet. Das Array hat allerdings nur ein Feld und ist deshalb schon zu Beginn auf ein Feld beschränkt worden.

Der SQL-Code ist hier zur besseren Übersicht noch einmal außerhalb der Makroformatierung aufgelistet:

```
SELECT "Datum"
FROM
   ( SELECT "a"."Datum",
      ( SELECT 'belegt' FROM "Reservierung"
         WHERE "Haus_ID" = 'inID'
         AND "a"."Datum" BETWEEN "Startdatum" AND "Enddatum" )
         AS "Haus"
   FROM "Ansicht_Datum" AS "a"
   WHERE "a"."Datum" >= CURRENT_DATE)
WHERE "Haus" IS NULL
```

Die innerste Abfrage aus der Tabelle "Reservierung" entspricht vom Inhalt her der Abfrage "Belegung" für. Sie ist durch die Auswahl des Fremdschlüssels für ein bestimmtes Haus nur auf ein Haus festgelegt. Als korrelierende Unterabfrage schreibt sie nur in alle Felder 'belegt', bei denen eben zu dem entsprechenden Datum der auf "Ansicht_Datum" bezogenen Abfrage ein Datumswert existiert, der zwischen den Datumswerten von "Startdatum" und "Enddatum" liegt. **`BETWEEN`** schließt hier die beiden Grenzdatumswerte mit ein.

Die beiden inneren Abfragen bilden zusammen die Datengrundlage für die äußere Abfrage, stellen also sozusagen eine Tabelle von Datumswerten zur Verfügung, aus denen ausgewählt werden kann. Die Datengrundlage ist dabei direkt auf Datumswerte beschränkt, die mindestens dem aktuellen Datum entsprechen, nicht aber in der Vergangenheit liegen.

Aus dieser Datengrundlage werden jetzt nur die Werte für "Datum" übernommen, bei denen noch kein Eintrag im Feld "Haus" verzeichnet ist. Zu diesen Zeitpunkten ist das Haus also noch frei.

In ungünstigen Fällen kann in so einer Liste bisher auch ein Datum stehen, zu dem überhaupt keine Buchung möglich ist. Dies ist in dem Screenshot beim 8.1.2015 für das Haus 1 der Fall, da vorher und nachher das Haus belegt ist. Hier wäre also gegebenenfalls noch Nachbesserungsbedarf.

Der SQL-Code wird an das Listenfeld über **`oFeld.ListSource`** weitergegeben. Anschließend muss das Feld mit **`oFeld.refresh`** neu eingelesen werden.

Nach der Auswahl des Startdatums wird über das Makro «Enddatum» das Listenfeld für das Enddatum mit entsprechenden Werten versorgt:

```
SUB Enddatum(oEvent AS OBJECT)
    DIM oFeldStart AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
```

```
    DIM inID AS INTEGER
    DIM stSql(0) AS STRING
    DIM stTeilsql AS STRING
    DIM stDatum AS STRING
    oFeldStart = oEvent.Source.Model
    oForm = oFeldStart.Parent
    oFeld = oForm.getByName("Enddatum")
    stDatum = oFeldStart.CurrentValue.Year & "-" &
       Right("0" & oFeldStart.CurrentValue.Month,2) & "-" &
       Right("0" & oFeldStart.CurrentValue.Day,2)
    inID = oForm.getByName("Haus_ID").CurrentValue
    IF ISEMPTY(stDatum) THEN
       msgbox "Zuerst muss ein Startdatum ausgesucht werden."
       EXIT SUB
    END IF
    stTeilsql = "( SELECT ""a"".""Datum"", ( SELECT 'belegt' FROM ""Reservierung""
       WHERE ""Haus_ID"" = '"+inID+"' AND ""a"".""Datum"" BETWEEN ""Startdatum""
       AND ""Enddatum"" ) AS ""Haus"" FROM ""Ansicht_Datum"" AS ""a""
       WHERE ""a"".""Datum"" > '"+stDatum+"')"
    stSql(0) = "SELECT ""Datum"" FROM "+stTeilsql+" WHERE  ""Datum"" <
       IFNULL((SELECT MIN(""Datum"") FROM "+stTeilsql+"
       WHERE ""Haus"" = 'belegt'),(SELECT MAX (""Datum"") FROM ""Ansicht_Datum""))"
    oFeld.ListSource = stSql
    oFeld.refresh
END SUB
```

Der Start ist für dieses Makro entsprechend dem Makro «Startdatum». Das Prinzip ist auch gleich. Es werden Werte aus Feldern ausgelesen, von denen dieses Feld abhängig ist. Es wird unter Berücksichtigung dieser Felder ein SQL-Code gebildet und anschließend wird das Listenfeld mit diesem Code versorgt und wieder neu eingelesen.

Das Feld «Startdatum» ist das auslösende Feld. Das Datum aus diesem Feld wird in Einzelteilen als Jahr, Monat und Tag gespeichert. Es muss also auch als Jahr, Monat und Tag ausgelesen werden. Beim Auslesen wird gleich darauf geachtet, dass ein Datum erstellt wird, das in der anschließend Abfrage brauchbar ist. Es muss also dem Format YYYY-MM-DD entsprechen. Entsprechend wird vor die Monatszahl und vor die Tageszahl einfach eine 0 gesetzt und anschließend über **`Right(Zahl,2)`** die letzten beiden Ziffern in das zu bildende Datum übernommen.

Wenn noch kein Datum in dem Feld «Startdatum» ausgesucht wurde ist **`stDatum`** anschließend leer. Dann erscheint einfach eine Meldung, dass zuerst ein Startdatum ausgesucht werden muss und das Makro wird mit **`EXIT SUB`** beendet.

Der SQL-Code ist hier in zwei Teile untergliedert. Zuerst wird eine Unterabfrage konstruiert, auf die die folgende Abfrage gleich doppelt zugreift. Auch hier zur besseren Übersicht zuerst die Unterabfrage **`stTeilsql`**, bereinigt um die Verdoppelung der Anführungszeichen, die äußere Klammerung und den Einbindungscode für die Variablen:

```
SELECT "a"."Datum",
   ( SELECT 'belegt' FROM "Reservierung"
      WHERE "Haus_ID" = 'inID'
         AND "a"."Datum" BETWEEN "Startdatum" AND "Enddatum" )
         AS "Haus"
      FROM "Ansicht_Datum" AS "a"
      WHERE "a"."Datum" > 'stDatum'
```

Ein Blick auf den Code für das Listenfeld «Startdatum» zeigt hier eine weitgehende identische Unterabfrage zu der dortigen Unterabfrage. Einzige Änderung ist hier die Beschränkung auf ein Datum, das auf jeden Fall nach dem bereits ausgewählten Startdatum liegen soll. Diese obige Abfrage bringt also eine Liste aller freien Datumswerte für das Haus, die nach dem Datum liegen, das bereits als Startdatum ausgewählt war. In der folgenden Abfrage wird dieser Abfrageteil einfach wie eine Tabelle stTeilsql aufgeführt.

```
SELECT "Datum"
FROM stTeilsql
```

```
WHERE "Datum" < IFNULL(
   (SELECT MIN("Datum") FROM stTeilsql WHERE ""Haus"" = 'belegt'),
   (SELECT MAX ("Datum") FROM "Ansicht_Datum")
   )
```

Es werden alle Datumswerte aus der Teilabfrage wiedergegeben. Sie werden allerdings dann begrenzt, wenn ein Wert in der Teilabfrage vorliegt, bei dem das Haus bereits belegt ist. Da in der Teilabfrage alle Datumswerte nach dem Wert aus Startdatum angezeigt werden, wird durch den kleinsten Datumswert, der bereits belegt ist, angezeigt, bis zu welchem Datumswert denn überhaupt eine Buchung in dem entsprechenden Haus möglich ist. Wenn in dem oben abgebildeten Formular für Haus 3 der 6.1.2015 als Startdatum gewählt würde, würden also in dem Listenfeld anschließend noch der 7.1., 8.1. und 9.1. zur Auswahl stehen. Wenn diese Zeiten nicht ausreichen muss eben ein anderes Haus oder ein anderes Startdatum gewählt werden.

Nach dem Löschen einer Buchung muss die Übersicht über die Buchungen in dem unteren Tabellenkontrollfeld aktualisiert werden. Über **Formular «BuchungLoeschen»** → **Ereignisse** → **Nach der Datensatzaktion** wird das folgende Makro gestartet:

```
SUB BuchungsAenderung
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Uebersicht")
    oForm.reload
END SUB
```

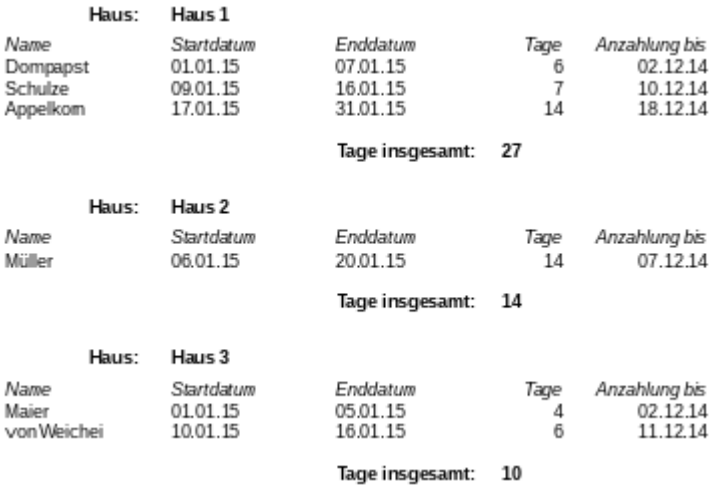
Über das geöffnete Dokument wird auf das Formular zugegriffen. Das Formular «Übersicht» wird neu geladen.

Nach dem Einfügen einer neuen Buchung ist diese in der Eingabemaske nicht mehr verfügbar. Ist die Buchung falsch, so muss sie über das Tabellenkontrollfeld zum Löschen einer Buchung verfügbar gemacht werden. Deshalb wird nach jeder erfolgten Buchung über **Formular «Formular»** → **Ereignisse** → **Nach der Datensatzaktion** das folgende Makro gestartet.

```
SUB NeueBuchung
    BuchungsAenderung
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("BuchungLoeschen")
    oForm.reload
END SUB
```

Die Funktionsweise ist gleich der des vorhergehenden Makros. Lediglich das Formular hat sich geändert. Prinzipiell könnte also auch mit einem einzigen Makro und der Weitergabe des Formularnamens über eine Variable gearbeitet werden.

Bericht



Haus: Haus 1

Name	Startdatum	Enddatum	Tage	Anzahlung bis
Dompapst	01.01.15	07.01.15	6	02.12.14
Schulze	09.01.15	16.01.15	7	10.12.14
Appelkorn	17.01.15	31.01.15	14	18.12.14

Tage insgesamt: 27

Haus: Haus 2

Name	Startdatum	Enddatum	Tage	Anzahlung bis
Müller	06.01.15	20.01.15	14	07.12.14

Tage insgesamt: 14

Haus: Haus 3

Name	Startdatum	Enddatum	Tage	Anzahlung bis
Maier	01.01.15	05.01.15	4	02.12.14
von Weichei	10.01.15	16.01.15	6	11.12.14

Tage insgesamt: 10

Der Bericht gibt lediglich eine Übersicht über die Belegung der Häuser zurück. Er ist gruppiert nach den Hausbezeichnungen. Im Gruppenfuß steht zusätzlich, wie viele Tage das Haus insgesamt belegt ist. Es gibt also so etwas wie eine Auslastung des Hauses in der angegebenen Zeitspanne wieder. Die Funktion für die Summierung ist eine der eingebauten Funktionen. Das Textfeld wurde aufgezogen und anschließend in **Eigenschaften** → **Daten** die folgenden Angaben gemacht:

- Datenfeld-Typ: Funktion
- Datenfeld: Tage
- Funktion: Summe
- Geltungsbereich: Gruppe: Haus

Rechnungen über Berichte erstellen

Diese Beispieldatenbank hat in der Hauptsache zum Ziel, eine mehrseitige Rechnungserstellung über Berichte zu erläutern. Die Tabellenbasis reicht dabei auch für ein einfaches Beispiel eines Geschäftsablaufes mit Kundenverwaltung aus. Allerdings wird in einem Formular lediglich die Auswahl der Kunden und die Zuordnung der Ware zu den Kunden aufgezeigt. Von dem Formular kann dann der Bericht direkt gestartet werden und die entsprechende Rechnung wird angezeigt.

Tabellen

Zur Rechnungserstellung werden die Tabellen "Ware", "Verkauf", "Rechnung" und "Kunde" benötigt.

In der Tabelle "Ware" werden folgende Felder definiert:

Feldname	***Feldtyp***	***Beschreibung***
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Ware	Text	Bezeichnung der Ware. **Feldeigenschaften** → **Eingabe erforderlich** → **Ja**
Preis	Dezimal	Preis der Ware. **Feldeigenschaften** → **Eingabe erforderlich** → **Ja**

Feldname	*Feldtyp*	*Beschreibung*
MWSt	Tiny Integer	Bezeichnung der Mehrwertsteuer, die für die entsprechende Ware erhoben wird. **Feldeigenschaften → Eingabe erforderlich → Ja**

Diese Tabellenkonstruktion arbeitet mit einem festgeschriebenen Preis. Preisänderungen würden es unmöglich machen, vergangene Rechnungen erneut korrekt auszudrucken. Es geht in diesem Beispiel in der Hauptsache um Berichte, nicht um eine komplette Warenverwaltung, die sowohl Ankauf als auch Verkauf beinhalten sollte und dann den Preis abhängig von Datumswerten separat erfassen würde.

Die Tabelle "Verkauf" ordnet die Waren einer entsprechenden Rechnung zu. Zusätzlich ermöglicht sie noch, entsprechende Stückzahlen zu den Waren fest zu legen.

In der Tabelle "Verkauf" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Anzahl	Tiny Integer	Wie viel von der bezeichneten Ware soll verkauft werden? In diesem Beispiel wird nicht mit großen Mengen gearbeitet. **Feldeigenschaften → Eingabe erforderlich → Ja**
Ware_ID	Integer	Fremdschlüssel aus der Tabelle "Ware". **Feldeigenschaften → Eingabe erforderlich → Ja**
Rechnung_ID	Integer	Fremdschlüssel aus der Tabelle "Rechnung". **Feldeigenschaften → Eingabe erforderlich → Ja**

Die Tabelle "Rechnung" dient als Verbindungselement zwischen dem Kunden und der verkauften Ware. Sie muss für die Erstellung einer Quittung mindestens ein Datum enthalten. Ohne so eine Tabelle würde jeder Kunde nicht nur seine waren, sondern auch die Waren aller vorhergehenden Kunden auf der Rechnung wieder finden.

In der Tabelle "Rechnung" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Datum	Tiny Integer	Wenigstens das aktuelle Datum sollte auf einer Rechnung oder Quittung vermerkt werden. **Feldeigenschaften → Eingabe erforderlich → Ja**
Kunde_ID	Integer	Fremdschlüssel aus der Tabelle "Kunde". Wird keine Rechnung sondern nur eine Quittung wie an einer Supermarktkasse erstellt, so kann dieses Feld auch leer bleiben.

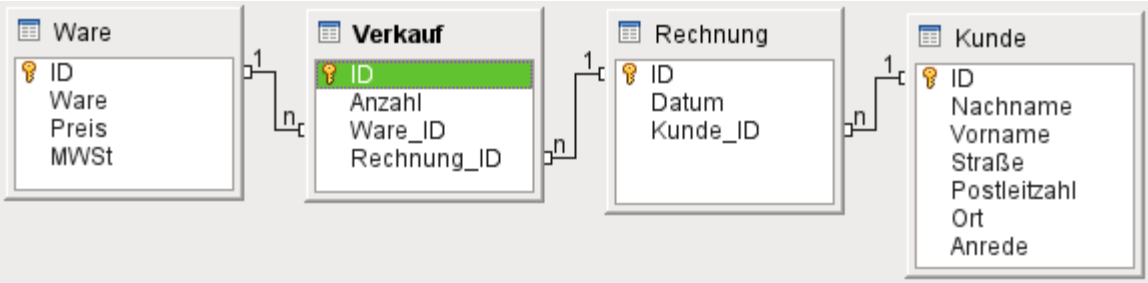
Die Tabelle "Kunde" zeigt eine Möglichkeit, wenigstens eine kleine Adressverwaltung mit der Rechnungserstellung zu verbinden.

In der Tabelle "Kunde" werden folgende Felder definiert:

Feldname	*Feldtyp*	*Beschreibung*
ID	Integer	Primärschlüssel der Tabelle. Der Primärschlüssel muss eindeutig sein. Das Feld ist als Auto-Wert-Feld gesetzt.
Nachname	Text	Wenigstens der Nachname sollte vom Kunden verzeichnet sein. **Feldeigenschaften → Eingabe erforderlich → Ja**

Feldname	*Feldtyp*	*Beschreibung*
Vorname	Text	Vorname des Kunden
Straße	Text	Straße und Hausnummer werden in einem Feld gespeichert.
Postleitzahl	Text	Hier ist die Länge des Textes auf 5 Zeichen beschränkt.
Ort	Text	Wohnort des Kunden
Anrede	Text	Anrede, die in den Rechnungen auftauchen soll.

Die Tabellen sind unter **Extras → Beziehungen** wie folgt miteinander verknüpft:



Zusätzlich zu den oben genannten Tabellen existiert noch eine Tabelle "Filter". Diese Tabelle hat in dieser Beispieldatenbank die Aufgabe, den Primärschlüssel der Tabelle "Rechnung" zu speichern, zu dem anschließend ein Ausdruck im Bericht erfolgen soll. Das entsprechende Feld ist deshalb ein Integer-Feld.

Des weiteren existiert eine Ansicht, die im folgenden Abschnitt «Abfragen» erklärt wird.

Abfragen

Ansichten eignen sich grundsätzlich besser als Abfragen zur Erstellung von Berichten. Bei Abfragen übernimmt der Report-Builder den Abfragecode und versucht anschließend damit seine Gruppierungen durch zu führen. Manchmal führt das dazu, dass der SQL-Code nicht verstanden wird und eine Gruppierung unmöglich ist. Manchmal wird ein Bericht überhaupt nicht ausgeführt, weil eine Alias-Zuweisung ignoriert wird.

Ansicht Bericht_Ware_gruppiert

Mit der Ansicht «Bericht_Ware_gruppiert» soll erreicht werden, dass bei gleichen Warennummern die Waren zusammengefasst werden und nicht mehrmals an unterschiedlichen Stellen in der Rechnung auftauchen.

```
SELECT
   MIN( "ID" ) "ID",
   SUM( "Anzahl" ) "Anzahl",
   "Rechnung_ID",
   "Ware_ID",
   "Datum",
   "Ware",
   "Preis",
   "Preis" * ( SELECT SUM( "Anzahl" ) FROM "Verkauf"
      WHERE "Ware_ID" = "a"."Ware_ID" AND "Rechnung_ID" =
      "a"."Rechnung_ID" )
      "Anzahl*Preis",
   ( SELECT SUM( "Preis" * "Anzahl" ) FROM "Ware", "Verkauf"
      WHERE "Verkauf"."Rechnung_ID" = "a"."Rechnung_ID"
      AND "Ware"."ID" = "Verkauf"."Ware_ID" )
```

```
      "Summe",
   ( SELECT "Anrede" || CHAR( 13 ) || "Vorname" || ' ' || "Nachname"
      || CHAR( 13 ) || "Straße" || CHAR( 13 ) || "Postleitzahl" || ' '
      || "Ort" FROM "Kunde", "Rechnung" WHERE "Kunde"."ID" =
      "Rechnung"."Kunde_ID" AND "Rechnung"."ID" = "a"."Rechnung_ID" )
      "Kunde",
   ( SELECT YEAR( "Datum" ) || '-' || "ID" FROM "Rechnung"
      WHERE "ID" = "a"."Rechnung_ID" )
      "Rechnungsnummer"
FROM "Verkauf" AS "a", "Rechnung", "Ware"
WHERE "Rechnung_ID" = IFNULL(
      ( SELECT "Integer" FROM "Filter" WHERE "ID" = TRUE ),
      "Rechnung_ID" )
   AND "Rechnung_ID" = "Rechnung"."ID"
   AND "Ware_ID" = "Ware"."ID"
GROUP BY "Rechnung_ID", "Ware_ID", "Datum", "Ware", "Preis"
```

Die Ansicht ist im obigen Beispiel einmal auf den direkt notwendigen Code zusammengestrichen worden. Alle Alias-Zuweisungen sind ohne **AS** formuliert. Die Tabellenzuweisungen zu entsprechenden Feldern sind nur dort erfolgt, wo Felder mit gleicher Benennung (**ID**) zu Verwechselung zwischen den Tabellen führen können.

Die Ansicht greift auf die Tabellen "Verkauf", "Rechnung" und "Ware" direkt zu. Da sich in der Abfrage Funktionen befinden müssen alle anderen Felder, die noch direkt in der Abfrage vorkommen, in der Liste der Gruppierungen erscheinen. Hier stehen neben "Rechnung_ID" und "Ware_ID" auch "Datum" und "Ware", die aber aus anderen Tabellen stammen. Da es für das Feld "Datum" und das Feld "Ware" aber in den anderen Tabellen keine gleichlautende Bezeichnung gibt kommt die HSQLDB auch ohne diese Bezeichnungen aus.

Aus dem Feld "ID" wird das Minimum berechnet. Dies soll lediglich bewirken, dass noch ein Primärschlüsselfeld aus der Tabelle "Verkauf" abgebildet wird. Es sollen ja gerade mehrere Datensätze dieser Tabelle zu einem Datensatz zusammen gefasst werden, sofern sie in Bezug auf "Rechnung_ID" und "Ware_ID" gleich sind. Das wäre nicht möglich, wenn mehrere "Verkauf"."ID" erscheinen.

Die Anzahl der Waren wird summiert, sofern es sich eben um Waren handelt, die die gleiche Waren_ID haben. Alle folgenden Felder werden direkt übernommen. Für Felder, die mit der Berechnung des Preises in Abhängigkeit von der Anzahl zusammen hängen wird auf eine korrelierende Unterabfrage zurück gegriffen. Es ist nicht möglich, direkt innerhalb der Grundabfrage noch einmal auf **SUM("Anzahl")** zuzugreifen.

Ebenfalls mit einer korrelierenden Unterabfrage wird die Gesamtsumme des Einkaufes berechnet. Dies könnte auch innerhalb des Berichtes einer entsprechenden Funktion überlassen werden.

Auch die Zusammenführung von mehreren Feldern zu einem Text mittels **||** führt zu Problemen bei Abfragen, die mit aggregierenden Funktionen (**MIN, MAX, SUMME**) arbeiten. Daher sind auch die Felder für die Aufnahme der Anschrift und die Zusammenstellung der Rechnungsnummer in korrelierenden Unterabfragen eingefügt. Bei der Anschrift wird über **CHAR(13)** jeweils ein Zeilenumbruch eingefügt. Wird dieser Zeilenumbruch nicht durchgeführt, so ist stattdessen auf eine Kombination von **CHAR(13) || CHAR(10)** zurück zu greifen, die unabhängig vom Betriebssystem zum Erfolg führt.

Ist in der Filtertabelle ein Filterwert eingetragen, so wird die darin benannte Primärschlüsselnummer aus der Tabelle "Rechnung" aufgelistet. Ohne einen Eintrag können auch alle bisher ausgestellten Rechnungen für den Ausdruck zusammen erscheinen.

Abfrage Verkauf_berechnet

Mit dieser Abfrage soll innerhalb des Formulars eine Eingabe ermöglicht werden, die gleichzeitig die Ergebnisse der Berechnungen liefert. So etwas wäre theoretisch auch mit einer einfachen Kombination von Tabellen in einer Abfrage möglich, sofern alle Primärschlüssel der beteiligten Tabellen enthalten sind. Leider aktualisiert aber Base diesen Abfragetyp nicht korrekt, wenn, wie in diesem Beispiel, über zwei Tabellen hinweg Berechnungen durchgeführt werden. Daher ist die Abfrage so aufgebaut, dass sie sich nur auf die Tabelle "Verkauf" bezieht und alle zusätzlichen Werte über korrelierende Unterabfragen hinzufügt.

```
SELECT "a".*,
   ( SELECT "Preis" * "a"."Anzahl" FROM "Ware"
      WHERE "ID" = "a"."Ware_ID" )
      AS "Anzahl*Preis",
   ( SELECT SUM( "Ware"."Preis" * "Verkauf"."Anzahl" ) FROM "Ware",
      "Verkauf" WHERE "Verkauf"."Rechnung_ID" = "a"."Rechnung_ID"
      AND "Ware"."ID" = "Verkauf"."Ware_ID" )
      AS "Summe",
   ( SELECT "Datum" FROM "Rechnung" WHERE "ID" = "a"."Rechnung_ID" )
      AS "Datum",
   ( SELECT "Ware" FROM "Ware" WHERE "ID" = "a"."Ware_ID" )
      AS "Ware",
   ( SELECT "Preis" FROM "Ware" WHERE "ID" = "a"."Ware_ID" )
      AS "Preis",
   ( SELECT "Kunde"."Anrede" || CHAR( 13 ) || CHAR( 10 ) ||
      "Kunde"."Vorname" || ' ' || "Kunde"."Nachname" || CHAR( 13 ) ||
      CHAR( 10 ) || "Kunde"."Straße" || CHAR( 13 ) || CHAR( 10 ) ||
      "Kunde"."Postleitzahl" || ' ' || "Kunde"."Ort" FROM "Kunde",
      "Rechnung" WHERE "Kunde"."ID" = "Rechnung"."Kunde_ID" AND
      "Rechnung"."ID" = "a"."Rechnung_ID" )
      AS "Kunde",
   ( SELECT YEAR( "Datum" ) || '-' || "ID"
      FROM "Rechnung" WHERE "ID" = "a"."Rechnung_ID" )
      AS "Rechnungsnummer"
FROM "Verkauf" AS "a"
```

Alle Felder aus der Tabelle "Verkauf" werden in der Abfrage dargestellt. Da der Tabelle der Alias "a" zugeordnet ist, muss auch bei dem Bezug auf alle Felder mit diesem Alias gearbeitet werden: **"a".*** . Ohne Alias würde ***** ausreichen.

Der Preis wird unter Berücksichtigung des aktuellen Datensatzes und der darin enthaltenen "Ware_ID" berechnet. Die Summe der Multiplikation von Anzahl und Preis bezieht sich auf die "Rechnung_ID" des aktuellen Datensatzes. Damit wird der Gesamtbetrag ausgegeben.

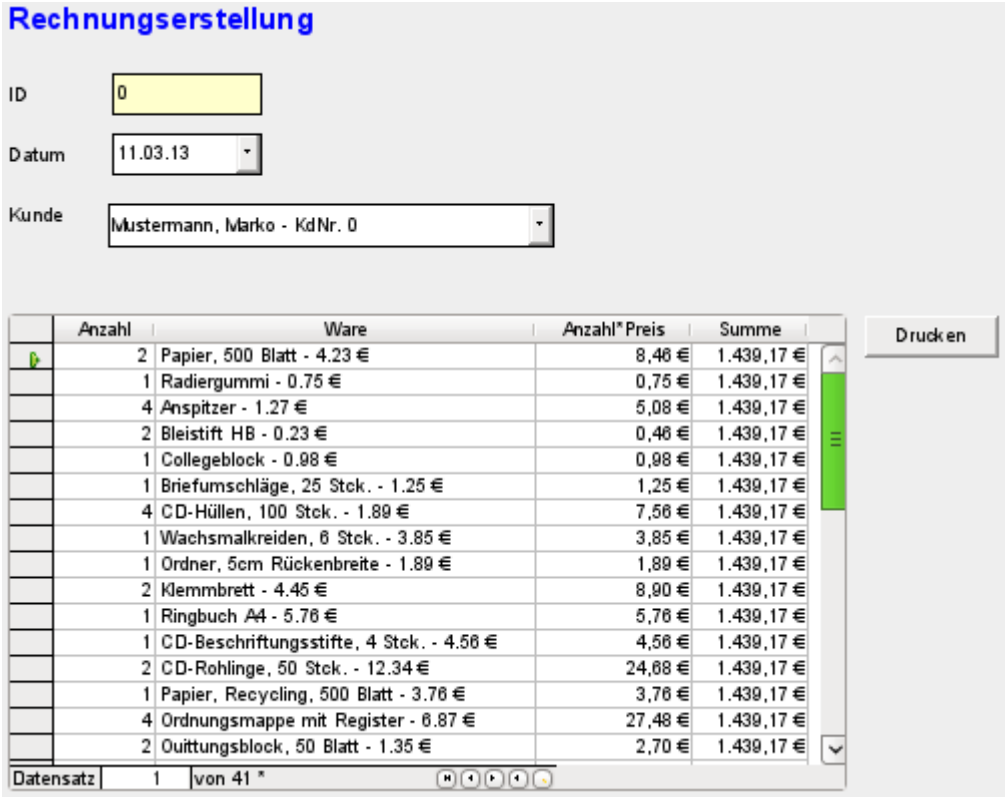
Datum, Warenbezeichnung und Preis werden durch korrelierende Unterabfragen hinzugefügt, damit die Abfrage editierbar bleibt. Ebenso wie in der vorhergehenden Ansicht die Anschrift des Kunden sowie die aus Jahreszahl und Rechnung_ID zusammengesetzte Rechnungsnummer.

Bei der Abfrage und auch bei der vorhergehenden Ansicht bleiben die Felder für die Adresse leer, wenn irgendein Eintrag fehlt. Falls beabsichtigt ist, auch eine Ausgabe nur mit der Nennung des Namens zu erzeugen (das Feld "Nachname" ist das einzige, was auf «Eingabe erforderlich» gestellt war), so müsste hier mit Einschüben über IFNULL nachgeholfen werden:

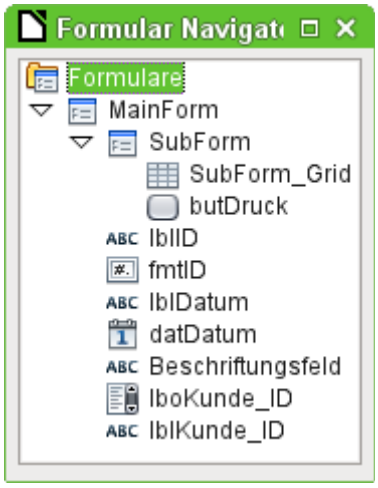
```
SELECT IFNULL("Kunde"."Anrede" || CHAR( 13 ) || CHAR( 10 ), '')|| ...
```

Hier wurde auf einen ausführlicheren Code verzichtet, genauso wie auf eine Eingabemaske für die Kundenverwaltung verzichtet wurde. Es sollte nur ein Formular erstellt werden, von dem aus eine Rechnungsausgabe über das Berichtsmodul erfolgen kann.

Formular



Das Formular enthält für das Aufrufen eines einzelnen Berichtes den Primärschlüssel der Tabelle "Rechnung" im Feld «ID». Dieser Primärschlüsselwert wird ausgelesen und über ein Makro in die Tabelle "Filter" übertragen. Anschließend wird in dem Makro noch der gewünschte Bericht gestartet.



Das Formular ist in ein Hauptformular mit Unterformular gegliedert. Das Hauptformular enthält alle Felder der Tabelle "Rechnung" als einfache Formularfelder. Das Feld für den Primärschlüssel "ID" ist schreibgeschützt und farblich auch als solches gekennzeichnet. Schließlich wird die "ID" ja über den Autowert erstellt.

Die Verbindung zum Unterformular, das sich auf die Abfrage «Verkauf_berechnet» bezieht, erfolgt über die "Rechnung_ID" der Abfrage mit der "ID" des Hauptformuars.

Das folgende Makro wird über die Schaltfläche «butDruck» **Ereignisse → Aktion ausführen** gestartet.

```
SUB Filtern_und_Drucken
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSQL AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("MainForm")
    oFeld = oForm.getByName("fmtID")
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "UPDATE ""Filter"" SET ""Integer"" = '"+oFeld.GetCurrentValue()+"' WHERE
        ""ID"" = TRUE"
    oSQL_Anweisung.executeUpdate(stSql)
    ThisDatabaseDocument.ReportDocuments.getByName(
        "Rechnung_gleiche_Ware_zusammengefasst_Kundenadressfeld").open
END SUB
```

Das Formular hat in diesem Beispiel den Namen "MainForm". Das Primärschlüsselfeld heißt "fmtID". Der Wert dieses Feldes wird ausgelesen und mit dem UPDATE-Befehl in die Tabelle "Filter" geschrieben. Anschließend wird der Bericht geöffnet und kann gedruckt werden.

Hier könnte durch den entsprechenden Einsatz von Makros auch direkt der Druck erfolgen. Zusätzlich könnte noch eine Sicherungskopie mit der Rechnungsnummer in einem beliebigen Verzeichnis abgelegt werden. Wie so etwas zu realisieren ist wird im Base-Handbuch ab Version 4.4 im Kapitel **Makros → Datenbankaufgaben mit Makros erweitert → Drucken aus Base heraus → Start, Formatierung, direkter Druck und Schließen des Berichtes** beschrieben.

Berichte

Die Wege, die zu einem Bericht wie dem hier abgebildeten führen, können als gesondertes Kapitel im Base-Handbuch nachgeschlagen werden: **Berichte → Beispiele für Berichte mit dem Report-Designer → Rechnungserstellung**. Ansonsten kann es auch helfen, mit einem Klick auf den Bericht und einem Blick ins Kontextmenü **Bearbeiten** zu wählen.

Büroshop

Norderstr. 17, 43219 Phantastica

Büroshop - Norderstr. 17 - 43219 Phantastica

Herrn
Marko Mustermann
Schloßallee 42
06741 Hintermberg

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
2	Papier, 500 Blatt	4,23 €	8,46 €
1	Radiergummi	0,75 €	0,75 €
4	Anspitzer	1,27 €	5,08 €
2	Bleistift HB	0,23 €	0,46 €
1	Collegeblock	0,98 €	0,98 €
1	Briefumschläge, 25 Stck.	1,25 €	1,25 €
4	CD-Hüllen, 100 Stck.	1,89 €	7,56 €
1	Wachsmalkreiden, 6 Stck.	3,85 €	3,85 €
1	Ordner, 5cm Rückenbreite	1,89 €	1,89 €
2	Klemmbrett	4,45 €	8,90 €
1	Ringbuch A4	5,76 €	5,76 €
1	CD-Beschriftungsstifte, 4 Stck.	4,56 €	4,56 €
2	CD-Rohlinge, 50 Stck.	12,34 €	24,68 €
1	Papier, Recycling, 500 Blatt	3,76 €	3,76 €
4	Ordnungsmappe mit Register	6,87 €	27,48 €
2	Quittungsblock, 50 Blatt	1,35 €	2,70 €
10	Rechnungsblock, 50 Blatt	3,15 €	31,50 €
2	Datumsstempel, einfach	18,50 €	37,00 €
2	Geldkassette, klein	12,00 €	24,00 €
24	Hängemappen, 25 Stck.	14,75 €	354,00 €
		Übertrag:	*554,62 €*

Büroshop

Norderstr. 17, 43219 Phantastica

Rechnungsnummer: 2013-0
Datum: 11.03.2013

Anzahl	Ware	Preis	Anzahl*Preis
		Übertrag:	*554,62 €*
4	Universaletiketten 70x32, 2700 Stck.	20,50 €	82,00 €
2	Universaletiketten Kleinpack 12x30, 700 Stck.	4,15 €	8,30 €
4	Druckerköpfe, schwarz	24,00 €	96,00 €
2	Druckerköpfe, color	26,30 €	52,60 €
6	Druckertintenpatronen, schwarz, 32ml	5,40 €	32,40 €
10	Druckertintenpatronen, color, 17ml	5,20 €	52,00 €
4	Toner Laserdrucker schwarz	65,89 €	263,56 €
2	Toner, Laserdrucker, color	78,89 €	157,78 €
4	Register für Ordner, A-Z	1,25 €	5,00 €
2	Heftstreifen, 25 Stck.	0,85 €	1,70 €
4	Schnellhefter Recyclingkarton	0,65 €	2,60 €
1	Papier, 500 Blatt, Recycling	4,15 €	4,15 €
1	Bleistifte, 10 Stck., versch. Stärken	4,85 €	4,85 €
2	Kugelschreiber	1,35 €	2,70 €
1	Wasserfarbkasten, 12 Farben	8,75 €	8,75 €
1	Aquarellkasten, 24 Farben	17,15 €	17,15 €
2	Aquarellpinsel, 3 Stck., versch. Stärken	8,34 €	16,68 €
1	Zeichenblock, A3, 20 Blatt	3,85 €	3,85 €
4	Schreibunterlage 50*70 cm	15,67 €	62,68 €
2	Tonpapier, div. Farben, 50*70 cm	0,45 €	0,90 €
10	Tonkarton, div. Farben, 50*70 cm	0,89 €	8,90 €
		Summe:	***1.439,17 €***

Kontoverbindung: Stadtsparkasse Phantastica - BLZ 123 456 01 - Konto 1029384756